DEFINITION AND VALIDATION OF A SOFTWARE METRIC BASED ON WORKLOAD

By

MICHAEL K. REYNOLDS

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1984

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July 2005

DEFINITION AND VALIDATION OF A SOFTWARE METRIC

BASED ON WORKLOAD

Thesis Approved:

M. H. Samadzadeh
Thesis Advisor

J. P. Chandler

B. E. Mayfield

A. Gordon Emslie
Dean of the Graduate College

PREFACE

Software "size" metrics play an important role in the field of measurement in software engineering. Size metrics help to quantify and estimate productivity, overall cost, progress, and process improvement. This thesis was a study to define a size metric based on the "workload" of the programming staff. In this context, the definition of workload is simply the total amount of code worked by the programming staff (code added, modified, and deleted in the implementation of the requirements for a version of a software product). The term "code" includes the source lines and the comment lines as well as the data files and script files required for complete implementation of the system requirements. The new metric, i.e., the Worked Lines of Code (WLOC) metric, was compared to other size metrics that have a good basis in the software industry already. Simple correlation analyses were applied to the data sets generated from four historical versions of a software project to compare the new metric to Source Lines of Code, Function Point Count, and Halstead Token Count.

The main objectives of this study were to define a new metric and compare it to a number of popular and established software metrics. Using software analysis tools from various vendors, size numbers were generated for four historical versions of a substantial application program from industry. In particular, data was generated for source lines of code (SLOC), Enhancement Function Point Count, and Halstead Token Count. The data for the metrics were collected from the four historical versions of the application using a

count utility designed and implemented to determine the lines of code added, modified, and deleted. The correlation study indicated strong relationships between the new metric and Function Point Count. The study found weak relationships with source lines of code and Halstead Token Count. Based on the data collected, the new metric was deemed a valid size measurement for software projects.

ACKNOWLEDGEMENTS

My sincere appreciation is extended:

To Dr. Mansur H. Samadzadeh for his continued support, guidance, and valuable insights throughout the development of this study.

To Mr. Milton Smith, Director, Software Engineering Directorate, Fire Support Software Engineering Center, Fort Sill, Oklahoma and Mr. Gary Meek, Vice President for Software Engineering Services and Program Manager for the Fire Support contract, Techrizon (formerly TELOS●OK, LLC) for allowing me access to the data and resources of the Forward Observer System (FOS) project to support this study.

To Mr. Phil Sperling, Principal Process Improvement Engineer, and Mr. Don Kesler, Quality Assurance and Configuration Management Specialist, Techrizon (formerly TELOS●OK, LLC) for their time, insights, and support in providing data to support this study.

To Ms. Tink Glenn, Director, Educational Outreach, Cameron University, for helping coordinate all the classroom space and times to help me meet my class schedule as a distance-learning student for this degree program.

To Ms. Mary S. Bradley, President, International Function Point Users Group, for reviewing and providing comments on the spreadsheet template I developed to count and track the function points presented in this study.

And finally, to my wife, Dawn, and my children, Bradley and Kaitlyn, I extend my heart-felt love and appreciation for their love and support in putting up with all my mood swings and frustrations as I worked to complete this study.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Background

Software Engineering is defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" [IEEE 93]. Metrics are collected and used by the software industry to quantify the development, operation, and maintenance of software and the software development process.

Computers are playing an ever-increasing role in almost every area of our lives. But computers and the software industry are also changing on a constant basis. Advances in hardware, software, graphics, development techniques, and languages mean that applications must be constantly rethought, retooled, and reengineered to maintain pace with these changes. This rapid evolution within the software industry makes management of the software development processes an extremely difficult task. This fact alone identifies the need to have some level of control or a set of standards to oversee the software development process and/or the software products.

Software measurement lends itself to establishing some degree of standardization for both the processes and products. As pointed out by Garmus and Herron, "the ability of an organization to effectively and efficiently manage data provides a true competitive advantage and adds value to the company's bottom line" [Garmus and Herron 96].

The overall cost associated with software development and software maintenance activities justify the need for standards with regard to software measurement. In 1980, Curtis [Curtis 80] indicated a need for scientific procedures to study the software development activities if programming was to be considered an engineering discipline. Curtis stated, "rigorous scientific procedures must be applied to studying the development of software systems if we are to transform programming into an engineering discipline". He advocated development of measurement techniques and determination of cause/effect relationships as the foundational approaches toward that goal.

DeMarco [DeMarco 82] stated, "you can't control what you can't measure". Hence, one of the basic needs for a software manager to be able to control the development of a software project is to measure the characteristics of the project. Measurements should help to improve the process and/or the quality of the product.

1.2 Objectives

Grady [Grady 87] [Grady 92] provided an answer to the question "why measure software?". He concluded that software measurement is used to provide a basis for estimates, track progress, determine complexity, understand quality, analyze the cause of defects, and validate best practices. Nearly everyone agrees that measuring software (either the development processes or the delivered products) aids management in making better decisions. Yet, there is no consensus within industry or academia as to what measurements are best as a set of standards for the software industry.

The objectives of this study are two-fold: 1) to define a new software measure based on the workload of the programming staff, and 2) to validate the new software

measure against other measures currently used in industry and academia as valid software measures. This new software measure (Worked Lines of Code) is basically another measurement of a project's size. It stems from the Configuration Management (CM) processes and products that have been employed by Techrizon (formerly TELOS●OK, LLC) for the past 15 years to track the lines of code added, modified, and deleted on any given software project. It has evolved from a manual count into an automated utility that can recognize various programming languages, scripting languages, and data file classifications.

Hartman and Austin [Hartman and Austin 93] looked at "changed lines of code" as the basis for a metric to define Maintenance Complexity. Their definition of "changed lines of code" is somewhat analogous to the idea of "Worked Line of Code", but their focus was on defining the complexity of the work being performed and not the validation of "changed lines of code" as a metric in its own right.

The remainder of this thesis report is organized as follows. Chapter II provides a brief history of software measurement. Chapter III provides a review of the software size measurements used as a part of this study. Chapter IV provides the evaluation of the WLOC metric based on correlation analyses with the other established metrics. Chapter V provides a summary, conclusion, and ideas for future work with regard to this new metric.

CHAPTER II

SOFTWARE MEASUREMENT

2.1 History

Software measurement has an established history dating back to the 1960s. Most agree that one of the earliest works in this field came from Rubey and Hartwick in 1968 [Rubey and Hartwick 68]. In this paper, Rubey and Hartwick define a quality model to address program quality for spacebourne software using quantitative measurements. At the heart of this model are seven "quality" attributes: correct calculations, correct logic, no interference between components, time and memory usage optimization, intelligibility, ease of modification, and ease of understanding and usage. These quality attributes correlate well with the goals of software engineering [Conte et al. 86]: efficiency, reliability, adaptability, maintainability, and usability.

Horst Zuse [Zuse 95] provides a web site with a very comprehensive history on the subject of Software Measurement. In his writings, he asserts that the "reasons for creating and inventing software measures" is mainly for the "development of reliable software".

Software reliability is probably the most critical attribute of a software system. If the software cannot perform its required functionality reliably, then its development costs (both time and effort) are wasted. Hence, software must be developed with a high degree of reliability at a reasonable cost. Several papers support this viewpoint; one of the most

significant is that of Boehm, Brown, and Lipow [Boehm et al. 76] as they attempted to define software quality. They developed and defined software quality based on several software characteristics: reliability, portability, efficiency, human engineering, testability, understandability, and modifiability. While these attributes are somewhat subjective, their use in establishing goals or guidelines to be followed during the software development process is a valuable tool for the software project manager.

2.2 Classification

As the discipline of software measurement evolved, measurements related to software became classified as process metrics or product metrics. As described by Conte, Dunsmore, and Shen [Conte et al. 86], process metrics deal with the development process and environment, while product metrics deal with the actual software products.

The classification of metrics is important because it helps to define the applicability and scope of the data being collected. One would not want to compare data that is unrelated, as the comparison would have no meaning. One would not want to collect data without having some basis for comparing that data either to the process involved or to the product produced.

2.3 Software Measurement at Techrizon (formerly TELOS●OK, LLC)

For Techrison (formerly TELOS●OK, LLC), software measurement has been a key focus area for process improvement activities. While software measurements were being gathered in the mid-1980's, they were not being fully utilized until efforts were made to implement a process improvement program for the company.

In 1991, an initial appraisal following the guidelines of the Capability Maturity Model® (CMM®) published by the Software Engineering Institute (SEI) at Carnegie Mellon University, Pittsburg, Pennsylvania, established Techrizon (formerly TELOS●OK, LLC) as a Level I company. Subsequent appraisals were conducted in 1994, 1997, and 2003 with ratings of Level III, Level IV, and Level V, respectively. The Level V assessment followed an updated CMM Integration (CMMI[SM]) model and a more rigorous Standard CMMI[SM] Appraisal Method for Process Improvement (SCAMPI[SM]).

One of the most significant achievements of the process improvement efforts was the ability to more accurately predict the size of the software projects during the initial stages of the development cycle. As stated by Smith and Sperling, "the organization's ability to accurately predict the size of the projects at the beginning of the development has improved 250 percent; as most companies realize, this ability is critical in estimating staffing and other resource needs" [Smith and Sperling 04].

CHAPTER III

SOFTWARE SIZE MEASUREMENTS

The best-known (and possibly most used) software measure is often referred to as "program size". There are several methods that are somewhat analogous in determining the program size: lines of code, token count, function point count, "reused" code, etc. Program size is often used as a basis for other types of metrics such as productivity, defect rate, and cost per LOC. Sheppard [Sheppard 93] suggested that program size is a good predictor of other indirect and more qualitative program characteristics such as reliability and maintainability.

3.1 Source Lines of Code (SLOC)

Source LOC or SLOC is the earliest software measure and the most basic approach used in determining program size [Park 92]. Nonetheless, there is tremendous disconnect on its base definition. Bailey and Basili [Bailey and Basili 81] asserted that LOC should be a "baseline metric to which all other metrics are compared". Their arguments suggested that LOC can be considered the "null hypothesis" for comparison studies and that any measure should perform better than the LOC measurement. The main problem with LOC as a metric is defining what is and what is not a line of code.

Several researchers have attempted to provide a stringent definition for LOC, but to date there is no single definition that is accepted across the software industry and academia.

Within the context of this study, the methodology of classifying, defining, and counting physical source statements as presented by Park [Park 92] was utilized to provide a uniform basis for our definition. Park [Park 92] also supports the idea that measurement is critical for software projects. It states, "size measures have a direct application to the planning, tracking, and estimating of software projects; they are used also to compute productivities, to normalize quality indicators, and to derive measures for memory utilization and test coverage". Indeed, the ability to properly size or estimate the size of a programming task is of utmost value to the software manager for planning, scheduling, and staffing the project.

3.2 Function Point Counts

While not a true size metric, function point counts provide a valid alternative to estimating program size. Albrecht [Albrecht 79] developed this method to estimate the "functionality" delivered by a program in terms of the number of function points (in lieu of estimating program size). His methodology estimates the functionality the software performs by counting the external inputs (EI) and output (EO), external queries (EQ), external interface files (EIF), and internal logical files (ILF) needed to implement the functionality of each subsystem constituting a software system. Albrecht claimed that "these factors are the outward manifestations of any application" [Albrecht 79]. It is possible with any development effort to collect these counts early in the requirement definition phase, independent of any programming language or implementation criteria.

Albrecht and Gaffney [Albrecht and Gaffney 83] validated this methodology against both the Halstead token count metrics and the basic Source LOC (SLOC) measurement. Based on this validation study, the correlation between function points and SLOC suggests that the function point count is a valid alternative to estimating the size of a program in terms of LOC, and it provides a good basis for determining effort and productivity for programming projects.

In 1986, the International Function Point Users Group (IFPUG) was founded to oversee the standardization of this metric and methodology [IFPUG 04]. One of the standardization methods employed by this group was the development and publication of the IFPUG Counting Practices Manual [IFPUG 01]. This manual provides the "rules" for counting the various inputs, outputs, data element types (DETs), record element types (RETs), and File Types Referenced (FTR). This manual also defines the standards for determining the complexity values (low, average, high) for these counts based on the number of DETs and FTRs or RETs employed by the various count items. Figures 1 through 3 show the complexity matrices for EI, EO, and ILF/ELF as defined by the IFPUG Counting Practices Manual. For an External Queries (EQ) count, either the EI or EO matrix is used based on whether there are more inputs (use EI matrix) or more outputs (use EO matrix) identified for the query. Garmus and Herron [Garmus and Herron 96] provide an excellent book describing the various counts and their complexities as well as providing helpful hints for the novice to get started utilizing function points.

The process for developing function point counts for an application follows a very basic set of steps.

Functional Complexity Matrix (EI)			
	DETs		
FTRs	**1-4**	**5-15**	**16 +**
< 2	Low	Low	Average
2	Low	Average	High
> 2	Average	High	High

Figure 1: Complexity Matrix for External Inputs (EI)

Functional Complexity Matrix (EO)			
	DETs		
FTRs	**1-5**	**6-19**	**20 +**
< 2	Low	Low	Average
2-3	Low	Average	High
> 3	Average	High	High

Figure 2: Complexity Matrix for External Outputs (EO)

Functional Complexity Matrix (EIF/ILF)			
	DETs		
RETs	**1-19**	**20-50**	**51 +**
< 2	Low	Low	Average
2-5	Low	Average	High
> 5	Average	High	High

Figure 3: Complexity Matrix for External/Internal Files (EIF/ILF)

1. Determine the specific Function Point count required.
2. Identify the boundary for the application.
3. Identify data functions and their complexities.
4. Identify transactional functions and their complexities.
5. Calculate the Unadjusted Function Point Count (weighted).
6. Determine the Value Adjustment Factors.
7. Calculate the Final Function Point Count.

To determine the type of Function Point Count (FPC) to use, one must examine the type of application under consideration. A Development Project FPC measures the functionality provided to the user with the first version of the application. An Enhancement Project FPC measures modifications to an existing application (functionality added, functionality deleted, and functionality changed). An Application FPC measures the functionality of an existing application in order to gather a baseline count for the application. For this study, the Enhancement Project FPC most closely matches the concept for the metric being proposed. Figure 4 provides an outline of the specific counts that must be gathered for the Enhancement Project FPC.

The Value Adjustment Factor (VAF) is a weighting factor used to determine the final Function Point counts based on fourteen General System Characteristics. These characteristics are evaluated based on their overall influence on the system. A scale from 0 (no influence) to 5 (strong influence) is used to classify the following characteristics.

1. Data Communications
2. Distributed Data Processing
3. Performance
4. Heavily Used Configuration
5. Transaction Rate
6. On-Line Data Entry
7. End-User Efficiency
8. On-Line Updates
9. Complex Processing
10. Reusability
11. Installation Ease
12. Operational Ease
13. Multiple Sites
14. Facilitate Change

Enhancement Function Point Counting Template

Function Points - Added	Low	Average	High	Total	
Inputs (EI)	0	0	0	0	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	0	0	0	0	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	0	0	0	0	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	0	0	0	0	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	0	0	0	0	(Low * 5 + Avg * 7 + High * 10)
	Added Function Points			0	

Function Points - Changed	Low	Average	High	Total	
Inputs (EI)	0	0	0	0	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	0	0	0	0	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	0	0	0	0	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	0	0	0	0	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	0	0	0	0	(Low * 5 + Avg * 7 + High * 10)
	Changed Function Points			0	

Function Points - Deleted	Low	Average	High	Total	
Inputs (EI)	0	0	0	0	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	0	0	0	0	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	0	0	0	0	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	0	0	0	0	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	0	0	0	0	(Low * 5 + Avg * 7 + High * 10)
	Deleted Function Points			0	

Figure 4: Enhancement Project Function Point Count Template

Figure 5 shows the requirements for gathering VAF before the enhancements are applied and after the enhancements are applied to the software project.

Once the various counts (Added, Changed, and Deleted) have been gathered and the VAF values (Before and After) have been calculated, the final Enhancement Function Point (EFP) can be calculated. EFP is calculated by the equation

$$EFP = (ADD + CHANGE) * VAF\ AFTER + DELETE * VAF\ BEFORE$$

where ADD = Function Points Added

CHANGE = Function Points Changed

VAF AFTER = Value Adjustment Factor after additions/changes

DELETE = Function Points Deleted

VAF BEFORE = Value Adjustment Factor before deletion

General System Characteristics	Before	After	Comment
1. Data Communication	0	0	
2. Distributed Data Processing	0	0	
3. Performace	0	0	
4. Heavily Used Configuration	0	0	
5. Transaction Rate	0	0	
6. On-Line Data Entry	0	0	
7. End-User Efficiency	0	0	
8. On-Line Updates	0	0	
9. Complex Processing	0	0	
10. Reusability	0	0	
11. Installation Ease	0	0	
12. Operational Ease	0	0	
13. Multiple Sites	0	0	
14. Facilitate Change	0	0	
Total Degree of Influence (TDI)	0	0	
Value Adjustment Factor (VAF)	0.65	0.65	(0.65 + (TDI * 0.01)

Figure 5: General System Characteristics Worksheet

3.3 Halstead Token Counts

Halstead [Halstead 77] introduced the concept of token count as a basis for a suite of metrics. His terminology for program size in tokens is "Program Length". Program size was characterized as a logarithmic function of the number of unique operators and operands within the program source code. His definition of size is derived from the program's source code using the following definitions:

n1 – the count of all unique operators

n2 – the count of all unique operands

N1 – the count of all operators

N2 – the count of all operands

N – the program length (N1 + N2)

N^ - the estimated program length ($n1*\log_2 n1 + n2*\log_2 n2$)

3.4 Reuse Code

Any software development project inherently lives with code reuse. Specifically, code that has been written for another software project, but either is used verbatim or is easily converted to handle the requirements of the new project, is what is meant by code reuse. Most programmers naturally attempt to reuse functions, features, and techniques that they have employed on previous projects. Conte, Dunsmore, and Shen [Conte et al. 86] claimed that "…in industry about 50 – 95% of what programmers do is modify existing code". Boehm [Boehm 81] accounted for code reuse in his COCOMO model. Bailey and Basili [Bailey and Basili 81] proposed a similar function to account for code reuse in their size estimation models. Each model claims that some level of attention must be given to the concept of "reuse" for the size estimation techniques to be of value.

Code reuse has been a salient focus in the processes and procedures utilized by Techrizon (formerly TELOS●OK, LLC) for software development. As early as 1994, Sodhi and Smith [Sodhi and Smith 94] defined the initial setup of software reuse requirements and a reuse repository that would be populated and used at the Fire Support Software Engineering Center (FSSEC) by Techrizon (formerly TELOS●OK, LLC). At Techrizon (formerly TELOS●OK, LLC), specific attention is given during requirement

analysis activities to identify requirements and functionality that can be incorporated into components that can be reused across several projects. Impacts and/or changes to existing reuse components are also identified during requirements analysis activities. In following this guideline, reuse code is logically separate and apart from the requirements specific to a software system being developed by Techrizon (formerly TELOS●OK, LLC). The only "code" that must be accounted for in the sizing of a project that utilizes one of these reuse components is the "code" that is written to provide the interface to the reuse component (conversion of data formats, simple parameter lists, etc.).

3.5 Worked Lines of Code (WLOC)

The concept of WLOC has evolved over the past 15 years in the process and product development at Techrizon (formerly TELOS●OK, LLC) from a simple counting utility of assembly language source statements using a "diff" utility, into a counting of various language and data file unique terms using a more robust count utility program.

The Configuration Management (CM) process (for the original assembly language products) required that data regarding "added", "modified", and "deleted" lines for both source and comments be identified as part of the Software Change Order (SCO). This data was originally collected manually by analyzing the output generated by using a "diff" command. An automated utility was created to aid the programming staff in the collection of this required data.

The original count utility took the output from the "diff" utility of a PDP 11/70 system and analyzed the "diff" output file against the original source file to determine if the line of data was "added", "modified", or "deleted". The syntax of the "diff" file

allowed for this classification by a simple analysis of the "diff" commands for "a" (added), "c" (changed), and "d" (deleted), respectively.

As the products (tactical system programs) evolved from the assembly world into the world of higher-order languages (mainly Ada), the classification of the "diff" output required better logic. The count utility program was modified to determine Ada lines of code. An Ada line of code was defined to be any source line of code ending in a semi-colon (Terminating Semi-Colons (TCS)) except when contained within textual strings and delimiters.

As implemented, the Ada line of code counting utility included counts for TSC, Comment Lines (CMT), Non-Comment Non-Blank (NCNB) lines, and Blank (BLNK) lines as part of its output data. This original count utility for Ada programs lacked in its ability to determine the "added", "modified", and "deleted" lines of code as required by the CM process.

As the CM process was further automated, the count utility was modified to determine a line of code based on the input syntax of the "language" under consideration. In this sense, the "language" could be Ada, Pascal, C, C++, Java, Unix Script files, or special Data files. Data files are categorized by the "commenting" characters (!, #, --, etc.) used within the files. The definition of the "language" of the file being analyzed allows for classification and identification of source lines and comment lines simply by defining the "commenting" character for the file under analysis.

The source and comment lines are gathered into unique files and then counted based on the following logic. If the line did not exist in the old file – added. If the line existed in the old file but not in the new file – deleted. If the line existed in both files –

modified. Tracking these line types in separate files evolved into a more accurate method of counting comment and source lines and provided for the requirements of the CM process to identify both source and comment lines that were “added”, “modified”, and “deleted”. The use of this utility across all projects ensured a unique, well-defined, and consistent method for collecting and reporting of the WLOC data.

One of the projects at Techrizon (formerly TELOS●OK, LLC) that has employed this WLOC concept is the Forward Observer System (FOS). FOS is a software package of products implemented in various programming languages (UNIX scripts, C, C++, and Ada) that provides automated digital message and data processing, data storage and recall, and communications capabilities to Field Artillery (FA) Fire Support (FS) personnel, FA Commanders (CDR), and FA Survey personnel. FOS provides three basic operational modes: Forward Observer (FO)/Fire Support Team (FIST), Fire Support Officer (FSO)/CDR, and Survey with each mode providing specific functionality to fulfill the selected role of the operator (FIST, CDR, Survey) in his/her tactical environment. The FOS software is integrated into several vehicle platforms to provide a tactical link to the FS network for locating, engaging, and tracking enemy targets. In addition, the FOS software provides a tactical link between the Tactical Internet (TI) and the FS network to allow TI assets direct access to FS assets for tactical Command and Control (C^2) activities.

CHAPTER IV

METRIC EVALUATION

The evaluation of any metric must start with a solid definition of the metric and a basic understanding of what the metric is and is not reporting. In this analysis, the metric under analysis is WLOC (worked lines of code) that basically is another metric that looks at a project's "size" attribute. Any project, big or small, requires a basic amount of work from the programming staff to analyze, design, code, unit test, and integrate the system. By focusing on the amount of code being "worked" (added, deleted, and modified) by the programming staff in performing these tasks, one should be able to more accurately determine the schedule requirements and the overall cost of the project.

4.1 Definition

Park [Park 92] provided a basic checklist to help define the attributes associated with "counting" source lines of code. Since WLOC is a "count" of the source lines of code that will be added, deleted, and modified, the checklist proposed by Park should allow for a concise definition of WLOC. Appendix C provides the details of this checklist as applied to WLOC.

4.2 Analysis of Data

In order to evaluate this metric against other "validated" metrics, the source code from the project had to be parsed to gather data to use for comparison. The gathering of this data was accomplished using "evaluation" copies of automated tools from various vendors and by developing Microsoft® Excel 2000 spreadsheets as required.

4.2.1 Source Lines of Code Data

Data specific to the Source Lines of Code (SLOC) was developed using an automated tool "SLOCCount" developed by David A. Wheeler [Wheeler 05]. This tool was chosen based on its ability to automatically detect the type of file (Ada, C, C++) being counted. Based on the documentation, this tool could easily be updated to handle the various data files associated with the FOS application to give an SLOC value that includes all the various components used in calculating WLOC. The output from running this tool against the various versions of FOS software is included as Appendix D. The SLOC values generated by this tool are shown in Figure 6.

Version	Ada	C	Script	C++	Total SLOC
V11	238292	6346	1700	474	246812
V12	242523	14833	897	475	258728
V7.0	264212	16207	2940	475	283834
V7.01	275653	66015	1235	475	343378

Figure 6: SLOC Count Summary

4.2.2 Function Point Count Data

Data specific to function point counts was developed using an Enhancement Project Function Point Count [IPFUG 01]. The Requirement Definition Documents

(RDDs) [RDD 99] [RDD 00] [RDD 03] [RDD 04] used to develop the FOS software were analyzed to determine appropriate function point counts. Appendix E contains the Microsoft® Excel 2000 spreadsheets showing the function point counts that were developed by a review of these requirement documents. The rules for counting and determining complexities as defining in the IFPUG Counting Practices Manual [IFPUG 04] (and as discussed by Garmus and Herron [Garmus and Herron 96]) were followed as closely as possible. In determining the Total Degree of Influent (TDI) for the FOS application, it was determined that both the "Before" and "After" values for the General System Characteristics were unaffected by the requirements as implemented; hence the Value Adjustment Factor (VAF) is the same for the "Before" and "After" influences. The final Adjusted Function Point counts are presented in Figure 7.

Version	Added	Changed	Deleted	VAF Before	VAF After	Adjusted FP
V11	3049	513	709	1.21	1.21	5168
V12	1575	576	0	1.21	1.21	2602
V7.0	1721	622	247	1.21	1.21	3134
V7.01	1963	520	215	1.21	1.21	3265

Figure 7: Enhancement Function Point Count Summary

4.2.3 Halstead Token Count Data

Data specific to the Halstead token counts [Halstead 77] was developed using software evaluation tools from Scientific Toolworks, Inc. [STI 05], called "Understand for Ada" and "Understand for C". These tools were used to parse the source code and a Perl script file (halstead.pl - downloaded from the Scientific Toolworks, Inc. web site) was used to generate the Halstead token counts for n1 (unique operators), n2 (unique operands), N1 (totals operators), N2 (total operands), N (length = N1 + N2), n

(vocabulary = n1 + n2), and N^ (estimated length = n1*$\log_2$ (n1) + n2*$\log_2$ (n2)). A small change was made to the Perl script file ("halstead.pl") in order to have the estimated length values (N^) displayed as part of the output from running the script. Partial output generated by the "halstead.pl" script for each software version is presented at Appendix F. A summary of the token counts generated by these tools is provided in Figure 8.

Version	n1	n2	N1	N2	N	n	N^
V11	55769	115318	568570	440981	1009551	171087	954068
V12	82702	183856	880353	689675	1570028	266558	1511423
V7.0	115976	240425	1127959	885383	2013342	356401	1980225
V7.01	116331	243059	1153900	904006	2057906	359390	2004937

Figure 8: Halstead Token Count Summary

4.2.4 Worked Lines of Code Data

The WLOC data was retrieved from an in-house developed Configuration Management Database (CMdb) utility that incorporated the line count utility that was previously described (see Section 3.5). The historical data for the FOS application was extracted and tabulated. Two totals were generated from this historical data: a total WLOC value for the FOS application and a WLOC value excluding the special data files associated with the FOS application. The data files were excluded in order to ensure a comparison of like data from the other automated utilities. These utilities do not possess the capability to properly count (as source code) the special data files that are developed as part of the FOS application. An example of the output provided from the CMdb utility is shown in Appendix G for Version 7.01 of the FOS application. The historical counts

are maintained for previous versions in the latest version's output data. A summary of the WLOC counts is provided in Figure 9.

Version	Ada	C	Script	Data	Total	w/o Data
V11	138574	1055	1384	243270	384283	141013
V12	53462	9346	843	33596	97247	63651
V7.0	72441	18732	2100	40650	133923	93273
V7.01	37616	64249	1558	33353	136776	103423

Figure 9: Worked Lines of Code Summary

4.3 Metric Evaluation and Discussion

To provide a validation of this new metric, a simple correlation study was conducted to search for relationships between the various data components extracted from the FOS source code (SLOC and Halstead counts) and requirement documentation (Function Point Counts). Figure 10 provides a table showing the calculated Pearson correlation coefficients (r) and the coefficient of determination (r^2) between the various data sets and the WLOC data. The question being addressed by this validation study is, "Does the WLOC metric/measurement provide a numerical characterization of the size attribute for software?". Our null hypothesis is that WLOC will be just as accurate with regards to system size as any of the other metrics considered within this study.

4.3.1 Source Lines of Code Comparison

Looking at the relationship between WLOC and SLOC, one finds a weak, linear, and independent relationship. Considering WLOC without Data (WLOC w/o Data), the data is inconclusive with regards to establishing a relationship for these data sets. These results were as expected. There is no significance in comparing the overall size of the

Data Sets	WLOC w/ Data r	WLOC w/ Data r^2	WLOC w/o Data r	WLOC w/o Data r^2
Source				
SLOC	-0.47	0.22	-0.06	0.004
SLOC Norm	0.96	0.93	0.97	0.94
SLOC Diff	0.98	0.96	0.92	0.85
Function Point				
AFP	0.99	0.98	0.94	0.89
AFP Norm	0.96	0.92	0.94	0.89
Halstead				
N (length)	-0.82	0.68	-0.48	0.23
N^ (estimated)	-0.82	0.68	-0.48	0.23

Figure 10: WLOC Correlation Coefficients for Data Sets

source code against the amount of code being added, modified, and deleted in the development of a software project. The final size (SLOC) of an application may or may not show growth in the subsequent versions of the product. If more requirements were removed than added, there may even be a decrease in the overall size of the final product. Thus, there is no clear indication that SLOC provides the same information as WLOC, and indeed, from the definition and understanding of WLOC, SLOC does not provide the same data.

Another value that was developed for SLOC was a "normalized" value. This value was calculated by determining the SLOC value from the function point count by multiplying the function point count by an approximation of the number of source statements required to code a single function point [Jones 95] (71 SLOC for Ada and 128 SLOC for C). This technique is referred to as "backfiring". In essence, the SLOC "Norm" value is an approximation of the amount of SLOC to be written based on the number of function points identified. For SLOC "Norm", there is a strong, linear, and

dependent relationship with WLOC and with WLOC without Data. This relationship was as expected based on the fact that the SLOC values came from the counts developed by the Function Point analysis.

A final value that was examined for SLOC was a simple difference in the final SLOC for each version (i.e., the amount of SLOC added from the previous version). This difference represents the "growth" of SLOC for each version of the product. A strong, linear, and dependent relationship was noted between SLOC "Diff" and WLOC. This result was somewhat unexpected because of the previous discussion concerning SLOC as a total. Had the difference been negative (i.e., more code deleted that added), this relationship would have been a weaker, possibly independent, relationship.

4.3.2 Function Point Count Comparison

Examining the data for Adjusted Function Point (AFP) counts as it relates to WLOC, one finds a strong, linear, and dependent relationship between the data sets to include the WLOC without Data values. Based on the definition of the Enhanced Function Point Count, a positive, strong correlation with WLOC was expected. Both data sets are trying to capture basically the same data – added, modified/changed, and deleted data.

Since SLOC was "normalized" based on function point counts (see Subsection 4.3.1), the same type of comparison was also conducted for the function point data. The Function Point counts were "normalized" by taking the SLOC data and dividing by the source lines per function point values [Jones 95] to achieve a "normalized" Function

Point count. The comparison of the AFP "Norm" to WLOC showed a strong, linear, and dependent relationship.

4.3.3 Halstead Token Count Comparison

For the Halstead token counts, comparisons were made with both the length token count ($N = N1 + N2$) and the estimated length token count ($n1*\log_2(n1) + n2*\log_2(n2)$). These token counts are basically equivalent to SLOC in that they represent the same type of data (i.e., they are proportional to the total source lines of code). Hence, expectations would indicate that the relationship should be similar to SLOC for the token counts under analysis. Indeed, for WLOC without Data, they values are almost identical (which seems intuitive since WLOC without Data should capture exactly the same information captured by SLOC). For WLOC, there was a somewhat strong, linear, yet independent relationship to the Halstead token counts.

From the study conducted by Albrecht and Gaffney [Albrecht and Gaffney 83], they showed that function point counting procedures are supported by the formulas presented by Halstead [Halstead 77] and that SLOC itself had a strong relationship with function point counts. This study does not contradict those findings.

CHAPTER V

SUMMARY, CONCLUSION, AND FUTURE WORK

5.1 Summary and Conclusion

The main focus of this thesis work was to examine a new metric (WLOC – worked lines of code) that is based on the workload of the programming staff. The workload of the programming staff includes specification, design (modeling), code, unit test activities, and integration test activities. A single "size" metric that captures this diversity of tasks is worth exploring.

For "size" measurements, the most important characteristic that adds validity to the data collected is utilizing a consistent definition over time. At Techrizon (formerly TELOS●OK, LLC), the WLOC data has been counted based on the same definition for over 15 years. The consistency of this definition allows Techrizon (formerly TELOS●OK, LLC) to make predictions against future work products, to determine productivity levels for the programming staff, and to validate cost and schedule requirements for future projects based on historical data.

Chapter I introduced the necessity for measuring software products and processes and the need for developing standards for these measurement techniques. Chapter II provided a history of software measurement and a classification of measurement. Chapter III reviewed the various "size" metrics that were reviewed as a part of this study. Although not a true measurement of "size", Function Point Count was presented as a part

of this study because of its validity as an alternative to estimating program size. Chapter IV presented the analysis of the various data components that were developed as part of this study. Correlation comparisons were made for various counts and most showed strong, linear, and dependent relationships with WLOC. The best comparison was with the Enhancement Function Point Count that has the same basic definition (added, changed, deleted) as WLOC.

In conclusion, WLOC was shown to be an effective alternative when measuring a programming project's size. The amount of code added, modified, and deleted is an effective method for determining the effort required (and thus the scheduling and resource requirements) for a programming task.

5.2 Future Work

Measurements that capture the total "size" of a programming task are both diverse in definition and unique in their respective "counting" rules. There is no industry standard for determining a programming project's size, yet there is strong agreement that measurements must be made to allow management to control the process and thus increase the quality of the product. With regards to WLOC, future analysis should consider data at the module level to achieve a more robust statistical analysis.

This study has generated interest in Function Points as a valid estimation technique for "size". A combination of Function Points (during requirement analysis and for initial size estimates using the "backfiring" technique), and WLOC (during code development and formal test activities) may provide a more robust process for tracking and monitoring progress and warrants further investigation.

REFERENCES

[Albrecht 79] A. J. Albrecht, "Measuring Application Development Productivity", in *GUIDE/SHARE: Proceedings of the IBM Applications Development Symposium*, pp. 83-92, Monterey, CA, October 1979.

[Albrecht and Gaffney 83] A. J. Albrecht and John E. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 6, pp. 639-648, November 1983.

[Bailey and Basili 81] J. W. Bailey and V. R. Basili, "A Meta-Model for Software Development Resource Expenditures", *Proceedings of the Fifth International Conference on Software Engineering*, pp. 107-116, San Diego, CA, March 1981.

[Boehm 81] Barry W. Boehm, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.

[Boehm et al. 76] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality", *Proceedings of the Second International Conference on Software Engineering*, pp. 592-605, San Francisco, CA, October 1976.

[Conte et al. 86] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*, Benjamin/Cummings Publishing Company, Menlo Park, CA, 1986.

[Curtis 80] Bill Curtis, "Measurement and Experimentation in Software Engineering", *Proceedings of the IEEE*, Vol. 68, No. 9, pp. 1144-1157, September 1980.

[DeMarco 82] Tom DeMarco, *Controlling Software Projects: Management, Measurement & Estimation*, Prentice Hall, Englewood Cliffs, NJ, 1982.

[Garmus and Herron 96] David Garmus and David Herron, *Measuring the Software Process: A Practical Guide to Functional Measurements*, Prentice Hall, Upper Saddle River, NJ, 1996.

[Grady 87] Robert B. Grady, "Measuring and Managing Software Maintenance", *IEEE Software*, Vol. 4, No. 9, pp. 35-45, September 1987.

[Grady 92] Robert B. Grady, *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, Englewood Cliffs, NJ, 1992.

[Halstead 77] M. H. Halstead, *Elements of Software Science*, Elsevier North-Holland, New York, NY, 1977.

[Hartman and Austin 93] Carl S. Hartman and Charles F. Austin, "Maintenance Productivity: Observations Based on an Experience in a Large System Environment", *Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research: Software Engineering - Volume 1*, pp. 138-170, Toronto, Ontario, Canada, October 1993.

[IEEE 93] IEEE Standard 610.12, *IEEE Standard Glossary of Software Engineering Terminology*, The Institute of Electrical and Electronic Engineers, Inc., Piscataway, NJ, 1993.

[IFPUG 01] International Function Point Users Group, *Function Point Counting Practices Manual*, Release 4.1, 2001.

[IFPUG 04] International Function Point Users Group, web reference: http://www.ifpug.org, last updated September 11, 2004, accessed March 2005.

[Jones 95] Capers Jones, "Backfiring: Converting Lines of Code to Function Points", *IEEE Computer*, Vol. 28, No. 11, pp. 87-88, November 1995.

[Park 92] Robert E. Park, "Software Size Measurement: A Framework for Counting Source Statements", *Technical Report – Software Engineering Institute*, SEI-92-TR-020, 220 pages, Pittsburgh, PA, May 1992.

[RDD 99] "Requirements Definition Document for Fire Support Systems Package 11", CECOM Software Engineering Center, Fort Sill, Oklahoma, 73503-6600, Revision F, 21 July 1999.

[RDD 00] "Requirements Definition Document for Fire Support Systems Version 12", CECOM Software Engineering Center, Fort Sill, Oklahoma, 73503-6600, Revision N/C, 15 February 2000.

[RDD 03] "Requirements Definition Document for Fire Support Version 7.0", CECOM Software Engineering Center, Fort Sill, Oklahoma, 73503-6600, Revision N/C, 5 February 2003.

[RDD 04] "Requirements Definition Document for Fire Support Version 7.01", CECOM Software Engineering Center, Fort Sill, Oklahoma, 73507-6600, Revision N/C, 10 May 2004.

[Rubey and Hartwick 68] R. J. Rubey and Dean Hartwick, "Quantitative Measurement of Program Quality", *Proceedings of the 1968 ACM 23rd National Conference*, pp. 671-677, Las Vegas, NV, August 1968.

[Shepperd 93] Martin Shepperd, *Software Engineering Metrics – Volume I: Measures and Validation*, McGraw Hill Book Company, International Series on Software Engineering, Berkshire, England, 1993.

[Smith and Sperling 04] Milton Smith and Phil Sperling, "The Fire Support Software Engineering Division Achieves CMMI Level 5", Crosstalk, Vol. 17, No.1, pp. 16-19, January 2004.

[Sohdi and Smith 94] Jag Sohdi and Milton Smith, "Marching Toward a Software Reuse Future", Crosstalk, Vol. 7, No. 9, pp. 21-27, September 1994.

[STI 05] "Understand for Ada" and "Understand for C", web reference: http://www.scitools.com, last updated March 2005, accessed March 2005.

[Wheeler 05] "SLOCCount", web reference: http://www.dwheeler.com/sloccount, last updated March 2005, accessed March 2005.

[Zuse 95] Horst Zuse, "A History of Software Measurement", web reference: http://irb.cs.tu-berlin.de/~zuse/metrics/3-hist.html, last updated September 1995, accessed August 2004.

APPENDICES

APPENDIX A

GLOSSARY

BLNK	Blank, a blank line in a program source file.
C^2	Command and Control, an operational mode that provides command and control of assets.
CDR	Commander, personnel in command of specific units and/or other personnel.
CM	Configuration Management, the process of controlling and managing software baselines.
CMM®	Capability Maturity Model®, a model of software engineering key practices devised by the SEI and used by the software industry.
CMMI^{SM}	Capability Maturity Model Integration^{SM}, a new model of software engineering key practices devised by the SEI that integrates software and system engineering.
CMT	Comment, a comment line in a program source file.
COCOMO	COnstructive COst MOdel, a model used to estimate the cost of a software project.
DET	Data Element Type, a user identifiable, non-repeated field or attribute maintained in the ILF or EIF. This includes foreign keys or special attributes attributed to the field [IFPUG 01].
EFP	Enhancement Function Point, a type of function point count used mainly for projects where functionality is being added, deleted, and modified.
EI	External Input, an elementary process that processes data or control information that comes from outside an application's boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system [IFPUG 01].

EIF	External Interface File, a user identifiable group of logically related data or control information referenced by an application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application counted. This means an EIF counted for an application must be in an ILF in another application [IFPUG 01].
EO	External Output, an elementary process that sends data or control information outside an application's boundary. The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information. The processing logic must contain at least one mathematical formula or calculation or create derived data. An external output may also maintain one or more ILFs and/or alter the behavior of the system [IFPUG 01].
EQ	External Inquery, an elementary process that sends data or control information outside an application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF or EIF. The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered [IPFUG 01].
FA	Field Artillery, units that provide artillery support to other units on the battlefield.
FIST	Fire Support Team, a group of personnel that act to provide fire support to other units on the battlefield.
FO	Forward Observer, personnel that locate and control artillery fire on enemy targets on the battlefield.
FOS	Forward Observer System, a tactical system used by the forward observer to track and control enemy targets.
FPC	Function Point Count, a count of the function points developed for an application's functional requirements.
FS	Fire Support, a role of the field artillery to provide supporting artillery fire to other units on the battlefield.
FSO	Fire Support Officer, an officer that coordinates fire support needs for other units on the battlefield.
FSSEC	Fire Support Software Engineering Center, an organization located at Fort Sill, Oklahoma that manages the production of software systems.

FTR	File Types Referenced, the total number of ILFs maintained, read, or referenced and the EIFs read or referenced by an input or output transaction (EI/EO) [IFPUG 01].
IEEE	Institute of Electrical and Electronics Engineers, an institution that defines and determines appropriate standards and methods for the electrical and electronics engineering disciplines.
IFPUG	International Function Point Users Group, a non-profit, member governed organization. The mission of IFPUG is to be a recognized leader in promoting and encouraging the effective management of application software development and maintenance activities through the use of Function Point Analysis and other software measurement techniques [IFPUG 04].
ILF	Internal Logical File, a user identifiable group of logically related data or control information maintained within the boundary of an application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted [IFPUG 01].
LOC	Lines of Code, a metric that identifies the size of a program source file.
NCNB	Non-Comment Non-Blank, a non-comment, non-blank line in a program source file.
PDP	Programmed Data Processor, a computer processor.
RDD	Requirement Definition Document, a document developed and controlled by Techrizon (formerly TELOS●OK, LLC) to capture the requirements and their system impacts.
RET	Record Element Type, a user identifiable subgroup of data elements contained within an ILF or ELF [IFPUG 01].
SCAMPI^{SM}	Standard CMMI^{SM} Appraisal Method for Process Improvement, an appraisal methodology devised by the SEI to standardize the appraisal process across the software industry.
SCO	Software Change Order, a formal request to initiate and control changes to a software baseline.
SEI	Software Engineering Institute, a federal research center that promotes the practices of software engineering to improve the quality of software systems.

TDI	Total Degree of Influence, a measure of the influence of fourteen general system characteristics on a software system's functionality.
TI	Tactical Internet, a network that mimics the Internet over a tactical radio networks.
TSC	Terminating Semi-Colon, a line in a program source file that ends with a semi-colon.
VAF	Value Adjustment Factor, a weighting of system characteristics to determine the adjustment value to be applied to unadjusted function point counts to reach a final function point count for an application.
WLOC	Worked Lines of Code, a metric based on the workload (LOC added, modified, and deleted) of a software project.

APPENDIX B

TRADEMARK INFORMATION

Capability Maturity Model® and CMM® are registered in the U.S. Patent and Trademark office.

Capability Maturity Model Integration, CMMI and SCAMPI are service marks of Carnegie Mellon University.

Microsoft® is a registered trademark of Microsoft Corporation.

APPENDIX C

WLOC DEFINITION CHECKLIST

This appendix contains a definition checklist [Park 92] that was developed for WLOC (worked lines of code) to provide a uniform basis for our definition of WLOC.

Definition Checklist for Source Statement Counts

Definition name: WORKED LINES OF CODE (basic definition - physical count)

Date: 3/15/05

Originator: MKR

Measurement unit:	**Physical source lines:** ☑						
	Logical source statements: ☐						
Statement type	**Definition** ☑	**Data array** ☐				**Includes**	**Excludes**
When a line or statement contains more than one type, classify it as the type with the highest precedence.							
1	Executable		**Order of precedence - >**		1	X	
2	Nonexecutable						
3	Declarations				2	X	
4	Compiler directives				3	X	
5	Comments						
6	On their own lines				4	X	
7	On lines with source code				5		X
8	Banners and nonblank spacers				6	X	
9	Blank (empty) comments				7	X	
10	Blank lines				8		X
11							
12							
How produced	**Definition** ☑	**Data array** ☐				**Includes**	**Excludes**
1	Programmed					X	
2	Generated with source code generators					X	
3	Converted with automated translators						X
4	Copied or reused without change						X
5	Modified					X	
6	Removed					X	
7							
8							
Origin	**Definition** ☑	**Data array** ☐				**Includes**	**Excludes**
1	New work; no prior existence					X	
2	Prior work; taken or adapted from						
3	A previous version, build, or release					X	
4	Commercial, off-the-shelf software (COTS), other than libraries						X
5	Government furnished software (GFS), other than reuse libraries						X
6	Another product						X
7	A vendor-supplied language support library (unmodified)						X
8	A vendor-supplied operating system or utility (unmodified)						X
9	A local or modified language support library or operating system						X
10	Other commercial library						X
11	A reuse library (software designed for reuse)						X
12	Other software component or library						X
13							
14							
Usage	**Definition** ☑	**Data array** ☐				**Includes**	**Excludes**
1	In or part of the primary product					X	
2	External to or in support of the primary product						X
3							

The terms in this checklist are defined and discussed in CMU/SEI-92-TR-20

Definition Checklist for Source Statement Counts

Definition name: WORKED LINES OF CODE
(basic definition - physical count)

	Delivery	Definition ☑	Data array ☐	Includes	Excludes
1	Delivered				
2	Delievered as source			X	
3	Delivered in compiled or execuatble for, but not as source				X
4	Not delivered				
5	Under configuration control				X
6	Not under configuration control				X
7					

	Functionality	Definition ☑	Data array ☐	Includes	Excludes
1	Operative			X	
2	Inoperative (dead, by-passed, unused, unreferenced, or unaccessed)				
3	Functional (intentional dead code, reactivated for special purposes)			X	
4	Nonfunctional (unintentionally present)			X	
5					
6					

	Replications	Definition ☑	Data array ☐	Includes	Excludes
1	Master source statements (originals)			X	
2	Physical replicates of master statements, stored in the master code				X
3	Copies inserted, instantiated, or expanded when compiling or linking				X
4	Postproduction replicates - as in distributed, redundant, or reparameterized systems				X
5					

	Development status	Definition ☑	Data array ☐	Includes	Excludes
	Each statement has one and only one status, usually that of its parent unit.				
1	Estimated or planned				X
2	Designed				X
3	Coded			X	
4	Unit test completed			X	
5	Integrated into components			X	
6	Test readiness review completed			X	
7	Software (CSCI) tests completed			X	
8	System tests completed			X	
9					
10					
11					

	Language	Definition ☑	Data array ☐	Includes	Excludes
	List each source language on a separate line.				
1		UNIX scripts, makefiles, special		X	
2	Job control languages	data files		X	
3					
4	Assembly languages				
5					
6	Third generation languages	Ada, C, C++		X	
7					
8	Fourth generation languages				
9					
10	Microcode				
11					

The terms in this checklist are defined and discussed in CMU/SEI-92-TR-20 Page 2

Definition Checklist for Source Statement Counts

Definition name: WORKED LINES OF CODE (basic definition - physical count)		Includes	Excludes
Clarifications (general) **Listed elements are assigned to**			
1 Nulls, contrinues, and no-ops **statement type - >**	1	X	
2 Empty statements (e.g., ";;" and lone semicolons on separate lines)	1	X	
3 Statements that instantiate generics	2	X	
4 Begin…end and {…} pairs used as executable statements	1	X	
5 Begin…end and {…} pairs that delimit (sub)program bodies	1	X	
6 Logical expressions used as test conditions			X
7 Expression evaluations used as subprogram arguments			X
8 End symbols that terminate executable statements	1	X	
9 End symbols that terminate declarations or (sub)program bodies	1	X	
10 Then, else, and otherwise symbols	1	X	
11 Keywords like procedure division, interface, and implementation			X
12 Labels (branching destinations) on lines by themselves	2	X	
13			
14			
15			
16			
Clarifications (language specific)			
Ada			
1 End symbols that terminate declarations or (sub)program bodies	1	X	
2 Block statements (e.g., begin…end)			X
3 With and use clauses	1	X	
4 When (the keyword preceding executable statements)			X
5 Exception (the keyword, used as a frame header)			X
6 Pragmas	2	X	
7			
8			
9			
Assembly			
1 Macro calls			
2 Macro expansion			
3			
4			
5			
6			
C and C++			
1 Null statements (e.g., ";" by itself to indicate an empty body)	1	X	
2 Expression statements (expressions terminated by semicolons)	1	X	
3 Expressions separated by semicolons, as in a "for" statement)			X
4 Block statements (e.g., {…} with no terminating semicolon)			X
5 "{", "}", or "};" on a line by itself when part of a declaration	2	X	
6 "{" or "}" on line by itself when part of an executable statement	1	X	
7 Conditionally compiled statements (#if, #ifdef, #ifndef)	2	X	
8 Preprocessor statements other than #if, #ifdef, and #ifndef	2	X	
9			
10			
11			
12			

Definition Checklist for Source Statement Counts

Definition name: WORKED LINES OF CODE (basic definition - physical count)		Includes	Excludes
CMS-2	**Listed elements are assigned to**		
1 Keywords like SYS-PROC and SYS-DD	**statement type - >**		
2			
3			
4			
5			
6			
7			
8			
9			
COBOL			
1 "PROCEDURE DIVISION", "END DECLARATIVES", etc.			
2			
3			
4			
5			
6			
7			
8			
9			
FORTRAN			
1 END statements			
2 Format statements			
3 Entry statements			
4			
5			
6			
7			
8			
JOVIAL			
1			
2			
3			
4			
5			
6			
7			
8			
Pascal			
1 Execuatble statements not terminated by semicolons			
2 Keywords like INTERFACE and IMPLEMENTATION]			
3 FORWARD declarations			
4			
5			
6			
7			
8			
9			

Definition Checklist for Source Statement Counts

Definition name: WORKED LINES OF CODE (basic definition - physical count)		Includes	Excludes
Listed elements are assigned to			
1 Data files with ! **statement type - >**	2	X	
2 Data files with #	2	X	
3 Data files with --	2	X	
4			
5			
6			
7			
8			
9			
10			
11			
12			

Summary of Statement Types

Executable statements

Executable statements cause runtime actions. They may be simple statements such as assignments, goto's, procedure calls, macro calls, returns, breaks, exits, stops, continues, nulls, no-ops, empty statements, and FORTRAN'S END. Or they may be structured or compound statements, such as conditional statements, repetitive statements, and "with" statements. Languages like Ada, C, C++, and Pascal have block statements [begin…end and {…}] that are classified as executable when used where other executable statements would be permitted. C and C++ define expressions as executable statements when they terminate with a semicolon, and C++ has a <declaration> statement that is executable.

Declarations

Declarations are nonexecutable program elements that affect an assembler's or compiler's interpretation of other program elements. They are used to name, define, and initialize; to specify internal and external interfaces; to assign ranges for bound checking; and to identify and bound modules and sections of code. Examples include declarations of names, numbers, constants, objects, types, subtypes, programs, subprograms, tasks, exceptions, packages, generics, macros, and deferred constants. Declarations also include renaming declarations, use clauses, and declarations that instantiate generics. Mandatory begin…end and {…} symbols that delimit bodies of programs and subprograms are integral parts of program and subprogram declarations. Language superstructure elements that establish boundaries for different sections of source code are also declarations. Examples include terms such as PROCEDURE DIVISION DATA DIVISION, DECLARATIVES, END DECLARATIVES, INTERFACE, IMPLEMENTATION, SYS-PROC, and SYS-DD. Declarations, in general, are never required by language specifications to initiate runtime actions, although some languages permit compile implement them that way.

Compiler directives

Compiler directives instruct compilers, preprocessors, or translators (but not runtime systems) to perform special actions. Some, such as Ada's pragma and COBOL's COPY, REPLACE, and USE, are integral parts of the source language. In other languages like C and C++, special symbols like # are used along with standardized keywords to direct preprocessor or compiler actions. Still other languages rely on nonstandardized methods supplied by compiler vendors. In these languages, directives are often designated by special symbols such as #, $, and {$}.

APPENDIX D

"SLOCCount" OUTPUT

This appendix contains the output files generated by the "SLOCCount" utility [WHEELER 05] from analyzing the historical versions of the FOS source code.

Version 10 Output

```
Creating filelist for v10
Categorizing files.
Finding a working MD5 command....
Found a working MD5 command.
Computing results.


SLOC   Directory    SLOC-by-Language (Sorted)
70669   v10             ada=69519,ansic=613,sh=537


Totals grouped by language (dominant language first):
ada:          69519 (98.37%)
ansic:          613 (0.87%)
sh:             537 (0.76%)


Total Physical Source Lines of Code (SLOC)                = 70,669
Development Effort Estimate, Person-Years (Person-Months) = 17.49 (209.85)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 1.59 (19.07)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 11.01
Total Estimated Cost to Develop                           = $ 2,362,284
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

Version 11 Output

```
Creating filelist for background_processing
Creating filelist for c_code
Creating filelist for cdmt
Creating filelist for communications
Creating filelist for components
Creating filelist for conversions
Creating filelist for data_base
Creating filelist for forward_observer_system
Creating filelist for hardware_interface
Creating filelist for library
Creating filelist for message_processing
Creating filelist for operator_interface
Creating filelist for scripts
Creating filelist for survey
Creating filelist for transmit_rules
Categorizing files.
Finding a working MD5 command....
Found a working MD5 command.
Computing results.

SLOC   Directory    SLOC-by-Language (Sorted)
60407   operator_interface      ada=60407
60008   conversions             ada=60008
34679   survey                  ada=34679
20597   message_processing      ada=20597
18389   communications          ada=18389
10124   data_base               ada=10124
9862    library                 ada=9658,ansic=204
8098    transmit_rules          ada=8098
7677    hardware_interface      ada=7677
6616    c_code                  ansic=6142,cpp=474
5496    background_processing   ada=5496
2817    components              ada=2817
894     cdmt                    sh=894
806     scripts                 sh=806
342     forward_observer_system ada=342

Totals grouped by language (dominant language first):
ada:         238292 (96.55%)
ansic:         6346 (2.57%)
sh:            1700 (0.69%)
cpp:            474 (0.19%)


Total Physical Source Lines of Code (SLOC)                = 246,812
Development Effort Estimate, Person-Years (Person-Months) = 65.02 (780.18)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.62 (31.40)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 24.84
Total Estimated Cost to Develop                           = $ 8,782,663
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

Version 12 Output

```
Creating filelist for background_processing
Creating filelist for c_code
Creating filelist for communications
Creating filelist for components
Creating filelist for conversions
Creating filelist for data_base
Creating filelist for forward_observer_system
Creating filelist for hardware_interface
Creating filelist for library
Creating filelist for master
Creating filelist for message_processing
Creating filelist for operator_interface
Creating filelist for scripts
Creating filelist for survey
Creating filelist for transmit_rules
Categorizing files.
Finding a working MD5 command....
Found a working MD5 command.
Computing results.

SLOC   Directory    SLOC-by-Language (Sorted)
63213   operator_interface      ada=63213
60011   conversions             ada=60011
34707   survey                  ada=34707
20649   message_processing      ada=20649
18388   communications          ada=18388
15308   c_code                  ansic=14833,cpp=475
10227   data_base               ada=10227
9757    library                 ada=9757
8781    hardware_interface      ada=8781
8122    transmit_rules          ada=8122
5621    background_processing   ada=5621
2610    components              ada=2610
820     master                  sh=820
437     forward_observer_system ada=437
77      scripts                 sh=77

Totals grouped by language (dominant language first):
ada:         242523 (93.74%)
ansic:        14833 (5.73%)
sh:             897 (0.35%)
cpp:            475 (0.18%)

Total Physical Source Lines of Code (SLOC)                = 258,728
Development Effort Estimate, Person-Years (Person-Months) = 68.31 (819.78)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.67 (32.00)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 25.62
Total Estimated Cost to Develop                           = $ 9,228,417
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

Version 7.0 Output

```
Creating filelist for background_processing
Creating filelist for c_code
Creating filelist for communications
Creating filelist for components
Creating filelist for conversions
Creating filelist for data_base
Creating filelist for forward_observer_system
Creating filelist for hardware_interface
Creating filelist for library
Creating filelist for master
Creating filelist for message_processing
Creating filelist for operator_interface
Creating filelist for scripts
Creating filelist for sensor_process
Creating filelist for survey
Creating filelist for transmit_rules
Categorizing files.
Finding a working MD5 command....
Found a working MD5 command.
Computing results.

SLOC   Directory    SLOC-by-Language (Sorted)
69184   operator_interface      ada=69184
65209   conversions             ada=65209
34699   survey                  ada=34699
23087   message_processing      ada=23087
21563   communications          ada=21563
16706   c_code                  ansic=16207,cpp=475,sh=24
12086   library                 ada=12086
10492   data_base               ada=10492
8632    transmit_rules          ada=8632
8257    hardware_interface      ada=8257
5973    background_processing   ada=5973
2824    master                  sh=2824
2700    components              ada=2700
1978    sensor_process          ada=1978
352     forward_observer_system ada=352
92      scripts                 sh=92

Totals grouped by language (dominant language first):
ada:         264212 (93.09%)
ansic:        16207 (5.71%)
sh:            2940 (1.04%)
cpp:            475 (0.17%)


Total Physical Source Lines of Code (SLOC)                = 283,834
Development Effort Estimate, Person-Years (Person-Months) = 75.29 (903.50)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.77 (33.20)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 27.21
Total Estimated Cost to Develop                           = $ 10,170,897
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

Version 7.01 Output

```
Creating filelist for background_processing
Creating filelist for c_code
Creating filelist for communications
Creating filelist for components
Creating filelist for conversions
Creating filelist for data_base
Creating filelist for forward_observer_system
Creating filelist for hardware_interface
Creating filelist for library
Creating filelist for master
Creating filelist for message_processing
Creating filelist for operator_interface
Creating filelist for project
Creating filelist for scripts
Creating filelist for sensor_process
Creating filelist for survey
Creating filelist for transmit_rules
Categorizing files.
Finding a working MD5 command....
Found a working MD5 command.
Computing results.

SLOC   Directory     SLOC-by-Language (Sorted)
73289   operator_interface      ada=73289
67078   conversions             ada=67078
66518   c_code                  ansic=66015,cpp=475,sh=28
35214   survey                  ada=35214
23422   message_processing      ada=23422
23105   communications          ada=23105
12470   data_base               ada=12470
12181   library                 ada=12181
8974    transmit_rules          ada=8974
8540    hardware_interface      ada=8540
6023    background_processing   ada=6023
2968    components              ada=2968
2021    sensor_process          ada=2021
1200    master                  sh=1200
368     forward_observer_system ada=368
7       scripts                 sh=7

Totals grouped by language (dominant language first):
ada:          275653 (80.28%)
ansic:         66015 (19.23%)
sh:             1235 (0.36%)
cpp:             475 (0.14%)

Total Physical Source Lines of Code (SLOC)                = 343,378
Development Effort Estimate, Person-Years (Person-Months) = 91.96 (1,103.50)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 2.99 (35.83)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 30.80
Total Estimated Cost to Develop                           = $ 12,422,320
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL
license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

APPENDIX E

FUNCTION POINT COUNT WORKSHEETS

This appendix contains the Microsoft® Excel worksheets that were generated from analyzing the requirement documents for the historical versions of the FOS software.

V11 Enhancement Function Point Counts

Function Points - Added	Low	Average	High	Total	
Inputs (EI)	95	54	5	531	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	100	75	9	838	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	60	56	7	446	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	43	45	10	901	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	20	19	10	333	(Low * 5 + Avg * 7 + High * 10)
	Added Function Points - FPA			3049	

Function Points - Changed	Low	Average	High	Total	
Inputs (EI)	10	4	8	94	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	10	4	2	74	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	5	5	4	59	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	10	7	5	215	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	10	3	0	71	(Low * 5 + Avg * 7 + High * 10)
	Changed Function Points - FPC			513	

Function Points - Deleted	Low	Average	High	Total	
Inputs (EI)	25	10	4	139	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	25	13	4	193	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	10	4	8	94	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	5	12	4	215	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	2	4	3	68	(Low * 5 + Avg * 7 + High * 10)
	Deleted Function Points - FPD			709	

General System Characteristics	Before	After	Comment
1. Data Communication	5	5	System supports 5 comms protocols/standards
2. Distributed Data Processing	4	4	Each node has unique processing reqmts
3. Performace	4	4	Performance requirements mandated
4. Heavily Used Configuration	4	4	CPU usage mandated
5. Transaction Rate	4	4	Performance requirements mandated
6. On-Line Data Entry	5	5	All data entered by operator or thru rcvd msgs
7. End-User Efficiency	5	5	Efficiency mandated
8. On-Line Updates	4	4	Data loss protection mandated
9. Complex Processing	3	3	Logic, Math, Security control
10. Reusability	5	5	Reuse mandated
11. Installation Ease	3	3	Procedures provided
12. Operational Ease	3	3	Education level of end-user
13. Multiple Sites	4	4	4 unique Hardware platforms
14. Facilitate Change	3	3	Design for maintenance mandated
Total Degree of Influence (TDI)	56	56	
Value Adjustment Factor (VAF)	1.21	1.21	(0.65 + (TDI * 0.01)
Enhancement Adjusted Function Points		5168	[(FPA + FPC) * VAFB] + (FPD * VAFA)

V12 Enhancement Function Point Counts

Function Points - Added	Low	Average	High	Total	
Inputs (EI)	39	58	8	397	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	39	58	8	502	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	13	23	0	131	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	8	25	6	396	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	3	9	7	148	(Low * 5 + Avg * 7 + High * 10)
	Added Function Points			1574	

Function Points - Changed	Low	Average	High	Total	
Inputs (EI)	17	20	0	131	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	17	20	0	168	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	3	15	0	69	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	3	13	0	151	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	3	6	0	57	(Low * 5 + Avg * 7 + High * 10)
	Changed Function Points			576	

Function Points - Deleted	Low	Average	High	Total	
Inputs (EI)	0	0	0	0	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	0	0	0	0	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	0	0	0	0	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	0	0	0	0	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	0	0	0	0	(Low * 5 + Avg * 7 + High * 10)
	Deleted Function Points			0	

General System Characteristics	Before	After	Comment
1. Data Communication	5	5	System supports 5 comms protocols/standards
2. Distributed Data Processing	4	4	Each node has unique processing reqmts
3. Performace	4	4	Performance requirements mandated
4. Heavily Used Configuration	4	4	CPU usage mandated
5. Transaction Rate	4	4	Performance requirements mandated
6. On-Line Data Entry	5	5	All data entered by operator or thru rcvd msgs
7. End-User Efficiency	5	5	Efficiency mandated
8. On-Line Updates	4	4	Data loss protection mandated
9. Complex Processing	3	3	Logic, Math, Security control
10. Reusability	5	5	Reuse mandated
11. Installation Ease	3	3	Procedures provided
12. Operational Ease	3	3	Education level of end-user
13. Multiple Sites	4	4	4 unique Hardware platforms
14. Facilitate Change	3	3	Design for maintenance mandated
Total Degree of Influence (TDI)	56	56	
Value Adjustment Factor (VAF)	1.21	1.21	(0.65 + (TDI * 0.01)
Enhancement Adjusted Function Points		2602	[(FPA + FPC) * VAFB] + (FPD * VAFA)

V7.0 Enhancement Function Point Counts

Function Points - Added	Low	Average	High	Total	
Inputs (EI)	43	63	11	447	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	43	63	11	564	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	16	14	0	104	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	20	14	12	460	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	4	8	7	146	(Low * 5 + Avg * 7 + High * 10)
	Added Function Points			1721	

Function Points - Changed	Low	Average	High	Total	
Inputs (EI)	12	22	4	148	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	14	24	5	211	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	4	3	3	42	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	7	5	2	129	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	2	6	4	92	(Low * 5 + Avg * 7 + High * 10)
	Changed Function Points			622	

Function Points - Deleted	Low	Average	High	Total	
Inputs (EI)	3	8	5	71	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	4	9	2	75	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	2	2	2	26	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	4	1	1	53	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	1	1	1	22	(Low * 5 + Avg * 7 + High * 10)
	Deleted Function Points			247	

General System Characteristics	Before	After	Comment
1. Data Communication	5	5	System supports 5 comms protocols/standards
2. Distributed Data Processing	4	4	Each node has unique processing reqmts
3. Performace	4	4	Performance requirements mandated
4. Heavily Used Configuration	4	4	CPU usage mandated
5. Transaction Rate	4	4	Performance requirements mandated
6. On-Line Data Entry	5	5	All data entered by operator or thru rcvd msgs
7. End-User Efficiency	5	5	Efficiency mandated
8. On-Line Updates	4	4	Data loss protection mandated
9. Complex Processing	3	3	Logic, Math, Security control
10. Reusability	5	5	Reuse mandated
11. Installation Ease	3	3	Procedures provided
12. Operational Ease	3	3	Education level of end-user
13. Multiple Sites	4	4	4 unique Hardware platforms
14. Facilitate Change	3	3	Design for maintenance mandated
Total Degree of Influence (TDI)	56	56	
Value Adjustment Factor (VAF)	1.21	1.21	(0.65 + (TDI * 0.01)
Enhancement Adjusted Function Points		3134	[(FPA + FPC) * VAFB] + (FPD * VAFA)

V7.01 Enhancement Function Point Counts

Function Points - Added	Low	Average	High	Total	
Inputs (EI)	65	62	5	473	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	65	62	5	605	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	12	15	3	114	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	25	25	15	650	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	5	8	4	121	(Low * 5 + Avg * 7 + High * 10)
	Added Function Points			1963	

Function Points - Changed	Low	Average	High	Total	
Inputs (EI)	11	26	3	155	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	18	25	3	218	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	3	3	3	39	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	2	2	2	64	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	2	2	2	44	(Low * 5 + Avg * 7 + High * 10)
	Changed Function Points			520	

Function Points - Deleted	Low	Average	High	Total	
Inputs (EI)	4	5	2	44	(Low * 3 + Avg * 4 + High * 6)
Outputs (EO)	4	3	2	45	(Low * 4 + Avg * 5 + High * 7)
Queries (EQ)	3	3	2	33	(Low * 3 + Avg * 4 + High * 6)
File Access (ILF)	3	2	2	71	(Low * 7 + Avg * 10 + High * 15)
Interface Files (EIF)	1	1	1	22	(Low * 5 + Avg * 7 + High * 10)
	Deleted Function Points			215	

General System Characteristics	Before	After	Comment
1. Data Communication	5	5	System supports 5 comms protocols/standards
2. Distributed Data Processing	4	4	Each node has unique processing reqmts
3. Performace	4	4	Performance requirements mandated
4. Heavily Used Configuration	4	4	CPU usage mandated
5. Transaction Rate	4	4	Performance requirements mandated
6. On-Line Data Entry	5	5	All data entered by operator or thru rcvd msgs
7. End-User Efficiency	5	5	Efficiency mandated
8. On-Line Updates	4	4	Data loss protection mandated
9. Complex Processing	3	3	Logic, Math, Security control
10. Reusability	5	5	Reuse mandated
11. Installation Ease	3	3	Procedures provided
12. Operational Ease	3	3	Education level of end-user
13. Multiple Sites	4	4	4 unique Hardware platforms
14. Facilitate Change	3	3	Design for maintenance mandated
Total Degree of Influence (TDI)	56	56	
Value Adjustment Factor (VAF)	1.21	1.21	(0.65 + (TDI * 0.01)
Enhancement Adjusted Function Points		3265	[(FPA + FPC) * VAFB] + (FPD * VAFA)

APPENDIX F

SELECTED HALSTEAD OUTPUT

This appendix contains abbreviated output files captured from the "Understand for Ada" and "Understand for C" tools [STI 05] generated by invoking a Perl script ("halstead.pl") to analyze the Halstead Token Count for the historical versions of the FOS software.

V10 Halstead Output							
Entity Name	**n1**	**n2**	**N1**	**N2**	**N**	**n**	**N^**
ct_imdm.ada	118	469	1973	1634	3607	587	4105
mr_chess.ada	33	94	347	245	592	127	782
mr_inept.ada	6	1	7	2	9	7	15
mr_pampr.ada	29	100	266	231	497	129	804
mr_prank.ada	22	68	166	139	305	90	511
mr_rover.ada	6	1	7	2	9	7	15
mr_task.ada	143	263	751	603	1354	406	2103
mr_umber.ada	31	58	181	138	319	89	492
mt_route.ada	22	69	200	168	368	91	519
mt_seria.ada	27	70	187	164	351	97	557
mt_xtask.ada	254	541	1344	1043	2387	795	4053
ct_error.ada	19	29	56	43	99	48	220
mr_ide.ada	26	50	179	130	309	76	404
mr_ream.ada	31	72	215	159	374	103	597
mr_ride.ada	21	13	68	42	110	34	140
mr_sack.ada	24	28	86	58	144	52	244
mr_valid.ada	24	52	312	187	499	76	406
mt_auth.ada	23	48	135	98	233	71	372
mt_ckath.ada	23	40	99	78	177	63	316
mt_gtlin.ada	21	31	65	50	115	52	245
mt_next.ada	40	71	251	178	429	111	557
mt_rsath.ada	24	81	223	196	419	105	623
mt_rsser.ada	29	69	178	150	328	98	561
mt_setli.ada	21	28	62	46	108	49	226
mt_upath.ada	36	95	445	362	807	131	810
mt_upmes.ada	21	29	69	53	122	50	232
dm.ada	24	18	45	31	76	42	142
dm_check.ada	286	386	2418	1711	4129	672	3435
dm_del.ada	152	266	1431	1041	2472	418	2250
dm_erase.ada	74	130	650	491	1141	204	1090
dm_link.ada	17	31	67	58	125	48	222
dm_next.ada	46	70	401	300	701	116	602
dm_read.ada	22	37	109	83	192	59	290
dm_sync.ada	9	6	12	8	20	15	43
dm_write.ada	280	403	2355	1653	4008	683	3347
ds_write.ada	139	346	1349	1031	2380	485	2810
io_close.ada	14	14	34	26	60	28	106
io_creat.ada	23	37	90	65	155	60	296
io_error.ada	10	6	13	8	21	16	48
io_file.ada	10	6	14	9	23	16	48
io_get.ada	11	8	20	15	35	19	62
io_len.ada	26	57	165	127	292	83	454
io_next.ada	21	11	45	29	74	32	130
io_open.ada	22	29	75	55	130	51	238
io_put.ada	12	9	22	16	38	21	71
io_read.ada	26	29	89	67	156	55	262
o							
o							
o							
Totals:	23169	46063	215631	167101	382732	69232	383083

V11 Halstead Output							
Entity Name	**n1**	**n2**	**N1**	**N2**	**N**	**n**	**N^**
service_manager.2.ada	309	281	937	642	1579	590	2535
service_manager.abort_print.2.ada	45	101	272	203	475	146	796
service_manager.assign_print_buffer.2.ada	27	34	146	92	238	61	300
service_manager.background_task_type.2.ada	55	88	201	136	337	143	628
service_manager.bulk_transmit.2.ada	941	2464	13712	10756	24468	3405	20586
service_manager.generate_file_image.2.ada	27	56	208	168	376	83	453
service_manager.generate_screen_image.2.ada	42	84	376	291	667	126	762
service_manager.get_form_header.2.ada	24	38	106	86	192	62	309
service_manager.get_service_request.2.ada	143	414	2593	1906	4499	557	3480
service_manager.print_altitude_form.2.ada	25	66	471	429	900	91	514
service_manager.print_arty_astro_form.2.ada	23	61	360	342	702	84	465
service_manager.print_azimuth_distance_form.2.ada	24	47	183	157	340	71	371
service_manager.print_buffer_line.2.ada	33	83	538	377	915	116	695
service_manager.print_coord_convert_form.2.ada	25	58	876	848	1724	83	455
service_manager.print_form.2.ada	150	227	1194	849	2043	377	1879
service_manager.print_gauss_kruger_datum_form.2.ada	23	47	168	144	312	70	365
service_manager.print_grid_conv_form.2.ada	23	45	152	134	286	68	351
service_manager.print_hasty_astro_form.2.ada	23	51	205	187	392	74	393
service_manager.print_intersection_form.2.ada	28	65	294	262	556	93	525
service_manager.print_opord.2.ada	21	38	95	78	173	59	291
service_manager.print_polaris_tab_form.2.ada	23	61	599	581	1180	84	465
service_manager.print_resection_form.2.ada	23	49	229	211	440	72	379
service_manager.print_star_id_form.2.ada	23	55	225	207	432	78	421
service_manager.print_traverse_form.2.ada	58	132	875	748	1623	190	1112
service_manager.print_triangulation_form.2.ada	58	146	1229	1087	2316	204	1222
service_manager.print_trig_trav_form.2.ada	23	45	156	138	294	68	351
service_manager.print_user_defined_datum_form.2.ada	25	69	1388	1360	2748	94	537
service_manager.print_utm_geo_datum_form.2.ada	27	58	326	302	628	85	467
service_manager.purge_transmit.2.ada	31	77	326	243	569	108	635
a220_config.2.ada	32	18	46	21	67	50	154
a220_config.generate_a220_file.2.ada	31	260	689	589	1278	291	2238
application_header.2.ada	171	454	2122	1773	3895	625	3882
communications.2.ada	780	1262	3987	3112	7099	2042	9845
communications.check_members.2.ada	40	104	708	526	1234	144	794
communications.check_nets.2.ada	27	55	306	257	563	82	445
communications.check_setup.2.ada	20	39	97	73	170	59	292
communications.control_type.2.ada	946	2367	9816	7704	17520	3313	19037
communications.generate_comms_files.2.ada	157	443	2099	1624	3723	600	3546
communications.initialize_channel.2.ada	131	319	2219	1567	3786	450	2567
communications.put_message.2.ada	267	818	4555	3614	8169	1085	6892
communications.send.2.ada	88	215	861	718	1579	303	1703
communications_definitions.2.ada	134	87	328	237	565	221	850
communications_services.2.ada	326	435	1133	865	1998	761	3552
compression.2.ada	75	118	328	289	617	193	901
debug_messages.2.ada	178	558	3775	3598	7373	736	4576
format_tff.to_db.2.ada	21	42	108	95	203	63	318
format_tff.to_tff.2.ada	20	43	109	96	205	63	319
o							
o							
o							
Totals:	55769	115318	568570	440981	1009551	171087	954068

V12 Halstead Output							
Entity Name	**n1**	**n2**	**N1**	**N2**	**N**	**n**	**N^**
bulk_transmit.2.ada	31	42	107	82	189	73	325
bulk_transmit.bulk_transmit.2.ada	959	2604	14707	11654	26361	3563	21847
bulk_transmit.purge_transmit.2.ada	35	79	369	266	635	114	676
cctt_print.2.ada	42	38	95	64	159	80	308
generate_print_files.2.ada	81	91	239	186	425	172	675
generate_print_files.generate_db_file_image.2.ada	30	58	193	149	342	88	486
generate_print_files.make_db_print_file.2.ada	33	71	220	159	379	104	602
generate_print_files.make_screen_print_file.2.ada	34	85	396	304	700	119	716
generate_print_files.total_pages_of_data.2.ada	24	39	93	70	163	63	316
print_file_queue.2.ada	119	105	242	166	408	224	848
print_file_task.print_file.2.ada	40	59	244	185	429	99	459
printer_interface.2.ada	112	130	331	219	550	242	1114
service_manager.2.ada	27	17	44	26	70	44	132
service_manager.abort_print.2.ada	39	79	172	142	314	118	585
service_manager.background_task_type.2.ada	30	32	59	40	99	62	209
service_manager.get_service_request.2.ada	64	149	381	301	682	213	1186
survey_forms.2.ada	277	272	812	577	1389	549	2301
survey_forms.get_form_header.2.ada	22	37	104	85	189	59	290
survey_forms.print_altitude_form.2.ada	26	81	499	451	950	107	635
survey_forms.print_arty_astro_form.2.ada	24	76	388	364	752	100	584
survey_forms.print_azimuth_distance_form.2.ada	25	60	208	177	385	85	470
survey_forms.print_coord_convert_form.2.ada	26	73	904	870	1774	99	573
survey_forms.print_form.2.ada	133	185	965	672	1637	318	1535
survey_forms.print_gauss_kruger_datum_form.2.ada	24	62	196	166	362	86	479
survey_forms.print_grid_conv_form.2.ada	24	60	180	156	336	84	464
survey_forms.print_hasty_astro_form.2.ada	24	66	233	209	442	90	508
survey_forms.print_intersection_form.2.ada	29	78	316	278	594	107	630
survey_forms.print_polaris_tab_form.2.ada	24	76	627	603	1230	100	584
survey_forms.print_resection_form.2.ada	24	64	257	233	490	88	494
survey_forms.print_star_id_form.2.ada	24	70	253	229	482	94	539
survey_forms.print_traverse_form.2.ada	57	146	891	758	1649	203	1221
survey_forms.print_triangulation_form.2.ada	57	160	1242	1094	2336	217	1332
survey_forms.print_trig_trav_form.2.ada	24	59	184	160	344	83	457
survey_forms.print_user_defined_datum_form.2.ada	26	83	1416	1382	2798	109	651
survey_forms.print_utm_geo_datum_form.2.ada	27	73	354	324	678	100	579
a220_config.2.ada	32	18	46	21	67	50	154
a220_config.generate_a220_file.2.ada	32	258	694	594	1288	290	2226
application_header.2.ada	171	436	2132	1779	3911	607	3719
communications.2.ada	903	1303	4192	3156	7348	2206	10416
communications.check_a220.2.ada	20	49	136	100	236	69	361
communications.check_adds.2.ada	116	337	1342	1100	2442	453	2769
communications.check_bf.2.ada	19	48	149	112	261	67	348
communications.check_cctt.2.ada	21	85	296	238	534	106	636
communications.check_fs.2.ada	26	101	322	256	578	127	794
communications.check_inc.2.ada	16	34	78	61	139	50	236
communications.check_lop.2.ada	56	138	544	404	948	194	1146
communications.check_members.2.ada	40	104	708	526	1234	144	794
o							
o							
o							
Totals	82702	183856	880353	689675	1570028	266558	1511423

V7.0 Halstead Output							
Entity Name	**n1**	**n2**	**N1**	**N2**	**N**	**n**	**N^**
bulk_transmit.2.ada	31	42	107	82	189	73	325
bulk_transmit.bulk_transmit.2.ada	973	2693	15281	12134	27415	3666	22572
bulk_transmit.purge_transmit.2.ada	35	79	369	266	635	114	676
cctt_print.2.ada	42	38	95	64	159	80	308
generate_print_files.2.ada	81	91	239	186	425	172	675
generate_print_files.generate_db_file_image.2.ada	30	58	190	146	336	88	486
generate_print_files.make_db_print_file.2.ada	33	70	218	157	375	103	595
generate_print_files.make_screen_print_file.2.ada	34	87	398	305	703	121	732
generate_print_files.total_pages_of_data.2.ada	24	39	93	70	163	63	316
print_file_queue.2.ada	119	105	242	166	408	224	848
print_file_task.print_file.2.ada	40	57	245	185	430	97	446
printer_interface.2.ada	112	130	334	221	555	242	1114
service_manager.2.ada	27	17	44	26	70	44	132
service_manager.abort_print.2.ada	39	79	172	142	314	118	585
service_manager.background_task_type.2.ada	30	32	59	40	99	62	209
service_manager.get_service_request.2.ada	64	158	419	338	757	222	1259
survey_forms.2.ada	277	272	814	577	1391	549	2301
survey_forms.get_form_header.2.ada	22	37	104	85	189	59	290
survey_forms.print_altitude_form.2.ada	26	81	499	451	950	107	635
survey_forms.print_arty_astro_form.2.ada	24	76	388	364	752	100	584
survey_forms.print_azimuth_distance_form.2.ada	25	60	208	177	385	85	470
survey_forms.print_coord_convert_form.2.ada	26	73	904	870	1774	99	573
survey_forms.print_form.2.ada	155	224	1115	782	1897	379	1865
survey_forms.print_gauss_kruger_datum_form.2.ada	24	62	196	166	362	86	479
survey_forms.print_grid_conv_form.2.ada	24	60	180	156	336	84	464
survey_forms.print_hasty_astro_form.2.ada	24	66	233	209	442	90	508
survey_forms.print_intersection_form.2.ada	29	78	316	278	594	107	630
survey_forms.print_polaris_tab_form.2.ada	24	76	627	603	1230	100	584
survey_forms.print_resection_form.2.ada	24	64	257	233	490	88	494
survey_forms.print_star_id_form.2.ada	24	70	253	229	482	94	539
survey_forms.print_traverse_form.2.ada	57	146	891	758	1649	203	1221
survey_forms.print_triangulation_form.2.ada	57	160	1242	1094	2336	217	1332
survey_forms.print_trig_trav_form.2.ada	24	59	184	160	344	83	457
survey_forms.print_user_defined_datum_form.2.ada	26	83	1416	1382	2798	109	651
survey_forms.print_utm_geo_datum_form.2.ada	27	73	354	324	678	100	579
a220_config.generate_a220_file.2.ada	32	258	694	594	1288	290	2226
application_header.2.ada	174	439	2154	1792	3946	613	3764
b220_config.generate_b220_file.2.ada	32	269	745	637	1382	301	2331
communications.2.ada	1027	1497	4940	3752	8692	2524	11864
communications.check_a220.2.ada	23	59	162	118	280	82	451
communications.check_adds.2.ada	116	339	1346	1104	2450	455	2785
communications.check_b220.2.ada	23	59	164	120	284	82	451
communications.check_cctt.2.ada	21	85	296	238	534	106	636
communications.check_clearance.2.ada	70	194	686	574	1260	264	1552
communications.check_fs.2.ada	27	109	358	282	640	136	865
communications.check_inc.2.ada	16	38	84	67	151	54	263
communications.check_lop.2.ada	56	138	544	404	948	194	1146
o							
o							
o							
Totals:	115976	240425	1127959	885383	2013342	356401	1980225

V7.01 Halstead Output							
Entity Name	**n1**	**n2**	**N1**	**N2**	**N**	**n**	**N^**
bulk_transmit.2.ada	31	42	106	81	187	73	325
bulk_transmit.bulk_transmit.2.ada	949	2639	15287	12107	27394	3588	22142
bulk_transmit.purge_transmit.2.ada	35	79	367	264	631	114	676
cctt_print.2.ada	42	38	95	64	159	80	308
generate_print_files.2.ada	81	91	239	186	425	172	675
generate_print_files.generate_db_file_image.2.ada	57	87	365	258	623	144	803
generate_print_files.make_db_print_file.2.ada	35	74	226	162	388	109	638
generate_print_files.make_screen_print_file.2.ada	34	87	398	305	703	121	732
generate_print_files.total_pages_of_data.2.ada	24	39	93	70	163	63	316
print_file_queue.2.ada	119	105	242	166	408	224	848
print_file_task.print_file.2.ada	40	57	245	185	430	97	446
printer_interface.2.ada	112	130	334	221	555	242	1114
service_manager.2.ada	27	17	44	26	70	44	132
service_manager.abort_print.2.ada	39	79	170	140	310	118	585
service_manager.background_task_type.2.ada	30	32	59	40	99	62	209
service_manager.get_service_request.2.ada	64	156	418	337	755	220	1243
survey_forms.2.ada	277	272	814	577	1391	549	2301
survey_forms.get_form_header.2.ada	22	37	104	85	189	59	290
survey_forms.print_altitude_form.2.ada	26	81	499	451	950	107	635
survey_forms.print_arty_astro_form.2.ada	24	76	388	364	752	100	584
survey_forms.print_azimuth_distance_form.2.ada	25	60	208	177	385	85	470
survey_forms.print_coord_convert_form.2.ada	26	73	904	870	1774	99	573
survey_forms.print_form.2.ada	155	224	1115	782	1897	379	1865
survey_forms.print_gauss_kruger_datum_form.2.ada	24	62	196	166	362	86	479
survey_forms.print_grid_conv_form.2.ada	24	60	180	156	336	84	464
survey_forms.print_hasty_astro_form.2.ada	24	66	233	209	442	90	508
survey_forms.print_intersection_form.2.ada	29	78	316	278	594	107	630
survey_forms.print_polaris_tab_form.2.ada	24	76	627	603	1230	100	584
survey_forms.print_resection_form.2.ada	24	64	257	233	490	88	494
survey_forms.print_star_id_form.2.ada	24	70	253	229	482	94	539
survey_forms.print_traverse_form.2.ada	57	146	891	758	1649	203	1221
survey_forms.print_triangulation_form.2.ada	57	160	1242	1094	2336	217	1332
survey_forms.print_trig_trav_form.2.ada	24	59	184	160	344	83	457
survey_forms.print_user_defined_datum_form.2.ada	26	83	1416	1382	2798	109	651
survey_forms.print_utm_geo_datum_form.2.ada	27	73	354	324	678	100	579
a220_config.generate_a220_file.2.ada	32	264	723	623	1346	296	2283
application_header.2.ada	173	464	2256	1870	4126	637	3981
c220_config.generate_c220_file.2.ada	34	293	845	720	1565	327	2573
communications.2.ada	1134	1695	5581	4271	9852	2829	13451
communications.check_a220.2.ada	23	59	162	118	280	82	451
communications.check_adds.2.ada	116	340	1348	1106	2454	456	2793
communications.check_c220.2.ada	23	59	164	120	284	82	451
communications.check_cctt.2.ada	21	85	296	238	534	106	636
communications.check_clearance.2.ada	70	198	718	598	1316	268	1585
communications.check_fs.2.ada	27	109	358	282	640	136	865
communications.check_hf.2.ada	12	10	25	17	42	22	76
communications.check_inc.2.ada	16	38	84	67	151	54	263
o							
o							
o							
Totals:	116331	243059	1153900	904006	2057906	359390	2004937

APPENDIX G

SELECTED "CMdb" OUTPUT

This appendix contains the WLOC (worked lines of code) data for the FOS software as captured from the "CMdb" utility developed by Techrizon (formerly TELOS•OK, LLC) to aid in managing the baseline system.

ADDED, MODIFIED, DELETED LINECOUNT BY LANGUAGE for: FOS													
	ADA			C			SCRIPT			DATA			Totals
WLOC as of v11.017	117,010	12,044	9,520	997	51	7	1,014	126	244	197,925	9,345	36,000	384,283
WLOC as of v12.008	32,595	11,502	9,365	9,305	37	4	200	243	400	31,674	974	948	97,247
WLOC as of v7.0.03	54,052	10,745	7,644	16,631	238	1,863	0	175	1,925	25,734	4,231	10,685	133,923
VERSION	ADD	MOD	DEL	ADD	MOD	DEL	ADD	MOD	DEL	ADD	MOD	DEL	WLOC
7.01.00	11,817	3,534	18,522	57,923	256	226	783	222	253	22,470	804	6,971	123,781
7.01.01	250	321	218	1,260	873	1,049	94	35	13	705	245	683	5,746
7.01.02	943	588	518	1,437	492	657	90	2	53	1,200	88	175	6,243
7.01.03	0	0	0	0	0	0	0	2	0	0	2	0	4
7.01.04	37	19	0	52	13	11	0	2	0	3	2	0	139
7.01.05	3	1	0	0	0	0	0	2	0	0	1	0	7
7.01.06	0	4	0	0	0	0	0	2	0	0	1	0	7
7.01.07	0	2	2	0	0	0	0	2	0	0	1	0	7
7.01.08	13	1	0	0	0	0	0	1	0	0	1	0	16
7.01.09	620	177	26	0	0	0	0	2	0	0	1	0	826
												Total to Date:	136,776

VITA

Michael K. Reynolds

Candidate for the Degree of

Master of Science

Thesis: DEFINITION AND VALIDATION OF A SOFTWARE METRIC BASED ON WORKLOAD

Major Field: Computer Science

Biographical:

Education: Graduated from Bixby High School, Bixby, Oklahoma in May 1980; received Bachelor of Science in Computer Science and Mathematics from Oklahoma State University in December 1984; completed the requirements for Master of Science at the Computer Science Department at Oklahoma State University in July 2005.

Experience: Employed since 1985 as a programmer, analyst, software engineer, system engineer, software project manager, and senior software design engineer with Techrizon, (formerly TELOS•OK, LLC, TELOS, TELOS Corporation, TELOS Federal Systems) working on tactical Fire Support systems (namely the Multiple Launch Rocket System (MLRS) Fire Direction System (FDS), the Fire Direction Data Manager (FDDM), and the Forward Observer System (FOS)) employed by the U. S. Army for command and control of battlefield assets.

Name: Michael K. Reynolds Date of Degree: July 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: DEFINITIONS AND VALIDATION OF A SOFTWARE METRIC BASED ON WORKLOAD

Pages in Study: 59 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: Software "size" metrics play an important role in the field of measurement in software engineering. Size metrics help to quantify and estimate productivity, overall cost, progress, and process improvement. This thesis was a study to define a size metric based on the "workload" of the programming staff. In this context, the definition of workload is simply the total amount of code worked by the programming staff (code added, modified, and deleted in the implementation of the requirements for a version of a software product). The term "code" includes the source lines and the comment lines as well as the data files and script files required for complete implementation of the system requirements. The new metric, i.e., the Worked Lines of Code (WLOC) metric, was compared to other size metrics that have a good basis in the software industry already. Simple correlation analyses were applied to the data sets generated from four historical versions of a software project to compare the new metric to Source Lines of Code, Function Point Count, and Halstead Token Count.

Findings and Conclusions: The main objectives of this study were to define a new metric and compare it to a number of popular and established software metrics. Using software analysis tools from various vendors, size numbers were generated for four historical versions of a substantial application program from industry. In particular, data was generated for source lines of code (SLOC), Enhancement Function Point Count, and Halstead Token Count. The data for the metrics were collected from the four historical versions of the application using a count utility designed and implemented to determine the lines of code added, modified, and deleted. The correlation study indicated strong relationships between the new metric and Function Point Count. The study found weak relationships with source lines of code and Halstead Token Count. Based on the data collected, the new metric was deemed a valid size measurement for software projects.

ADVISOR'S APPROVAL: M. H. Samadzadeh