Recognized as an
American National Standard (ANSI)

IEEE Std 1045-1992

IEEE Standard for Software Productivity Metrics

Sponsor

Software Engineering Standards Subcommittee of the Technical Committee on Software Engineering of the IEEE Computer Society

Approved September 17, 1992

IEEE Standards Board

Approved March 22, 1993

American National Standards Institute

Abstract: A consistent way to measure the elements that go into computing software productivity is defined. Software productivity metrics terminology are given to ensure an understanding of measurement data for both source code and document production. Although this standard prescribes measurements to characterize the software process, it does not establish software productivity norms, nor does it recommend productivity measurements as a method to evaluate software projects or software developers. This standard does not measure the quality of software. This standard does not claim to improve productivity, only to measure it. The goal of this standard is for a better understanding of the software process, which may lend insight to improving it.
Keywords: attribute, primitive, productivity ratio, source statement, staff-hour

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Print: ISBN 1-55937-258-3, SH15651
PDF: ISBN 0-7381-0408-6, SS15651

Introduction

(This introduction is not a part of IEEE Std 1045-1992, IEEE Standard for Software Productivity Metrics.)

This introduction is intended to provide the reader with some background into the rationale used to develop the standard. This information is being provided to aid in the understanding and usage of the standard. The introduction is nonbinding.

This standard defines a framework for measuring and reporting software productivity. It focuses on definitions of how to measure software productivity and what to report when giving productivity results. It is meant for those who want to measure the productivity of the software process in order to create code and documentation products.

Past software productivity metrics have not proven as useful as desired to provide insight into the software process. Although there is an accumulation of more than 20 years of data, consistent productivity indicators for software development have not emerged from this information. The problem is not as much the fault of the metrics being used as it is the inaccuracy and incompleteness of the data being collected.

The definition of productivity states that it is the ratio of a unit of output to a unit of input used to produce the output. For this relationship to be useful for software, the data used in it must be accurate and complete. For instance, reported software productivity of 5000 lines of source code per year leaves many questions unanswered, *What is a line of source code? How long, in work hours, was the year? What activities were included? Whose effort was counted?*

Interpreting productivity based on a single number leaves much unknown about the process being measured. Without knowing the scope and characteristics of the process measured, or the precision of the data used in the calculations, the resulting productivity values are inconclusive.

The goal of this standard is to build a foundation to accurately measure software productivity. This is done through a set of precisely defined units of measure. However, not all software processes lend themselves to precise measurement. Software development is a new and rapidly evolving field, and it is strongly influenced by the variability of the people who build the software. In those situations where precise measurement definitions are not possible, this standard requests that descriptions of the processes used and the measurements taken be done in a specified format.

The intention of the standard is to formalize the presentation of productivity data so that it is useful to anyone wishing to improve the software process. This standard is not an end in itself. Instead, it is the beginning of increased precision in collecting and reporting software productivity data. The hope is that this will lead to an improved understanding of the software development process and to improved productivity metrics.

Participants

At the time this standard was completed, the Software Productivity Metrics Working Group had the following membership:

Robert N. Sulgrove, *Chair*
Eleanor Antreassian, *Past Chair*
Christine H. Smith, *Co-chair*
Nicholas L. Marselos, *Editor*

Bakul Banerjee
Stephen E. Blake
Thomas P. Bowen
David N. Card
Thomas J. Carlton
Deborah Caswell
Mike Demshki
Sherman Eagles
Ruth S. Euler
Michael Evangelist
Al Freund
James T. Fritsch
John E. Gaffney, Jr.
Stuart Glickman
Lennor Gresham
Stuart Jeans
Robert W. Judge
Lawrence King
Thomas M. Kurihara
Arnold W. Kwong
Fred Lau
Chi Yun Lin
Denis C. Meredith
Lois J. Morton
Andrew Najberg
Jainendra K. Navlakha
Dennis E. Nickle
Richard Reese
Julian Roberts
Brian Sakai
Sylvia Shiroyama
Paul Stevens
Wolfgang B. Strigel
Leonard L. Tripp
Scott A. Whitmire
Ron Willis
Paul Wolfgang

Contributors

The following individuals also contributed to the development of the standard:

William W. Agresti
Lowell Jay Arthur
Jeff A. Aune
Victor R. Basili
Mordechai Ben-Menachem
Victor G. Berecz
Robert C. Birss
Bob Bisschoff
Barry Boehm
George Bozoki
Fred Burke
Neva Carlson
Sally Cheung
Rutherford Cooke
Charles D’Argenio
James B. Dolkas
Carl Einar Dragstedt
Christof Ebert
Violet Foldes
Andrew S. Fortunak
Robert Fraley
Jack Fried
Jean A. Gilmore
Clell Gladson
J. G. Glynn
Robert Grady
Dan Grigore
Amal Gupta
Nash Hair
H. D. Hall
Jack Harrington
Warren Harrison
Bruce Healton
Francis B. Herr
Herman Hess
Geoffrey W. Higgin
John W. Horch
David Hurst
Randall W. Jensen
Bud Jones
T. Capers Jones
Bill Junk
Motti Y. Klein
Phil Kolton
Steven E. Kreutzer
Walt Kutz
Robert L. Lanphar, Jr.
F. C. Lim
Michael Lyu
Andy Mahindru
Jukka Marijarvi
Phillip C. Marriott
Roger J. Martin
Robert F. Martini
Joseph F. Mathews
Bruce Millar
James Miller
Robert C. Natale
Stephen R. Neuendorf
Ken O’Brien
Randy Paddock
Bruce Parker
Bud W. Pezet
Wes Philp
Robert M. Poston
Lawrence H. Putnam
Donald J. Reifer
Niall Ross
Vince Rupolo
Norman F. Schneidewind
Roger Scholten
David J. Schultz
Suzanne E. Schwab
Carl Seddio
Al Serna
Jean Shaffer
Vincent Y. Shen
Josef Sherif
David M. Siefert
Vijaya K. Srivastava
Edwin J. Summers
David Swinney
Robert C. Tausworthe
C. L. Troyanowski
Dolores Wallace
Richard Werling
John Westergaard
Clyde E. Willis
William Wong
Weider Yu

Sponsoring Organizations

The following persons were on the balloting committee that approved this standard for submission to the IEEE Standards Board:

Contents

IEEE Standard for Software Productivity Metrics

1. Overview

This standard describes the data collection process and calculations for measuring software productivity. This standard is divided into eight clauses. Clause 1 provides the scope of this standard. Clause 2 lists references to other standards that are useful in applying this standard. Clause 3 provides an abbreviated set of definitions and acronyms defined more fully in the standard, but provided here as a quick reference. Clause 4 provides an introduction to the standard's measurement approach. Clause 5 describes data collection for measuring output. Clause 6 describes data collection for measuring input. Clause 7 describes productivity results represented by ratios of outputs to inputs. Clause 8 provides a method to capture the characteristics of the software process to better understand their effects on productivity results.

This standard also contains four annexes. These provide additional information for understanding and using this standard, but they are not part of the standard. Annex A is a data collection form for recording quantitative metrics data defined in this standard, and Annex B is a data collection form for recording the characteristics data defined in this standard. Annex C contains a bibliography of some of the many references used in developing this standard. Annex D describes a counting method for the relationships defined in the standard.

1.1 Scope

Measuring software productivity is similar to measuring other forms of productivity; that is, it is measured as the ratio of units of output divided by units of input. The inputs consist of the effort expended to produce a given output. For software, the tangible outputs are source code and documentation. Of course the product of software is a process, an algorithm that drives a computer to do work. Unfortunately, understanding of the functionality incorporated into a software product is still rudimentary. Only by understanding what can be measured will it be possible to understand the essence of the software process.

Toward this end, this document standardizes software productivity metrics terminology to ensure an understanding of measurement data for both code and documentation production. It defines a set of units to measure the output products and input effort.

The lowest level of measurement defined in this standard is called a *primitive*. The output primitives measured are software source statements, documentation pages, and, optionally, function points. The input primitives measure the efforts of those developing the software products. The capacity and capability of automated support tools are not directly measured by this standard, but are indirectly measured by the improvements in the productivity of the people who use them.

This standard prescribes measurements to characterize the software process, and in doing so gives insight for improving it. This standard does not establish software productivity norms, nor does it recommend productivity measurements as a method to evaluate software projects or software developers.

Although the overall value of a product cannot be separated from the quality of what is produced, this standard does not measure the quality of software. The issue of measuring quality is beyond the scope of this standard, because it covers a different aspect of the software process. Nevertheless, productivity metrics should be interpreted in the context of the overall quality of the product. It is left to the user of this standard to look to other standards covering quality metrics for that information.

The definition of productivity as the ratio of product to effort was selected in this standard because it is more stable than value relationships like product to monetary units or production to sales. The effects of inflation on the value of monetary units, or the caprice of the marketplace on sales, cause these measures to vary unpredictably. As a result, they do not offer a stable measure over time. However, the quantity of the product and the effort in hours expended to produce it are more consistent measures, and over time these measures will provide a better gauge of software productivity. Users may wish to translate productivity into monetary equivalents, but results shall be reported in the units specified in this standard.

The metrics in this standard apply equally well to new development and to the enhancement or maintenance of an existing software product. Subsequent releases or changes to a released or delivered software product should be viewed as a new product for the purpose of applying these metrics.

This standard defines a consistent way to measure the elements that go into computing software productivity. A consistent measurement process will lead to a better understanding of the software development process, and a better understanding will lead to improvement. This standard does not claim to improve productivity, only to measure it.

1.2 Terminology

The words *shall*, *must*, and words in the imperative form identify the mandatory (essential) material within this standard. The words *should* and *may* identify optional (conditional) material. As with other IEEE Software Engineering Standards, the terminology in this document is based on the IEEE Std 610.12-1990.[1] To avoid inconsistency when the glossary is revised, the definitions are not repeated in this document. New terms and modified definitions as applied in this standard are included in clause 3.

1.3 Audience

This standard should be of interest to software development managers, software productivity and process assessment personnel, quality assurance personnel, software engineering improvement personnel, and researchers.

2. References

This standard shall be used in conjunction with the following publications:

IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology (ANSI).

IEEE Std 1074-1991, IEEE Standard for Developing Software Life Cycle Processes (ANSI).

[1]Information on references can be found in clause 2.

3. Definitions

This clause contains key terms as they are used in this standard.

3.1 activities: Events in the software life cycle for which effort data is collected and reported.

3.2 added source statements: The count of source statements that were created specifically for the software product.

3.3 attributes: Measurable characteristics of a primitive.

3.4 characteristics: Those inherent factors of software development that may have a significant impact on productivity.

3.5 comment source statements: Source statements that provide information to people reading the software source code and are ignored by the compiler.

3.6 compiler directive source statements: Source statements that define macros, or labels, or direct the compiler to insert external source statements (for example, an *include* statement), or direct conditional compilation, or are not described by one of the other type attributes.

3.7 data declaration source statements: Source statements that reserve or initialize memory at compilation time.

3.8 deleted source statements: Source statements that are removed or modified from an existing software product as a new product is constructed.

3.9 delivered source statements: Source statements that are incorporated into the product delivered to the customer.

3.10 developed source statements: Source statements that are newly created for, added to, or modified for a software product.

3.11 direct staff-hour: The amount of effort directly expended in creating a specific output product.

3.12 document page count: The total number of nonblank pages contained in a hard-copy document.

3.13 document screen count: The total number of page images for electronically displayed documents.

3.14 documents: Hard copy, screen images, text, and graphics used to convey information to people.

3.15 executable source statements: Source statements that direct the actions of the computer at run time.

3.16 function point: A measure of the delivered software functionality.

3.17 granularity: The depth or level of detail at which data is collected.

3.18 incremental productivity: The productivity computed periodically during development.

3.19 input primitive: The effort to develop software products, expressed in units of staff-hours.

3.20 logical source statements (LSS): Source statements that measure software instructions independently of the physical format in which they appear.

3.21 modified source statements: Original source statements that have been changed.

3.22 new source statements: The sum of the added and modified source statements.

3.23 nondelivered source statements: Source statements that are developed in support of the final product, but not delivered to the customer.

3.24 nondeveloped source statements: Existing source statements that are reused or deleted.

3.25 origin attribute: The classification of software as either developed or nondeveloped.

3.26 original source statements: Source statements that are obtained from an external product.

3.27 output primitives: Primitives that include source statements, function points, and documents.

3.28 page size: The edge-to-edge dimensions of hard-copy documents, or the average characters per line and the number of lines per screen for electronically displayed documents.

3.29 physical source statements (PSS): Source statements that measure the quantity of software in lines of code.

3.30 primitive: The lowest level for which data is collected.

3.31 productivity ratio: The relationship of an output primitive to its corresponding input primitive.

3.32 reused source statements: Unmodified source statements obtained for the product from an external source.

3.33 source statements (SS): The encoded logic of the software product.

3.34 staff-hour: An hour of effort expended by a member of the staff.

3.35 support staff-hour: An hour of effort expended by a member of the staff who does not directly define or create the software product, but acts to assist those who do.

3.36 tokens: The content of a document as characterized by words, ideograms, and graphics.

3.37 type attributes: The classification of each source statement as either executable, data declaration, compiler directive, or comment.

3.38 usage attributes: The classification of software as delivered to a user of the final product, or as nondelivered when created only to support the development process.

4. Software productivity metrics

This standard defines measurement primitives for computing software productivity. A primitive is the lowest level for which data is collected. The output primitives for software are source statements and, optionally, function points. For documentation, the output primitive is the page. The input primitive is the staff-hour.

Primitives are categorized by attributes. An attribute is a measurable characteristic of a primitive. This standard requires that all attributes be measured for each primitive. For example, the source statement primitive may be either new or reused, and it may also be either delivered or nondelivered. Each of these attributes is a different variation of the primitive. Thus, a new delivered source statement differs from a new nondelivered source statement. By using this scheme of primitives and attributes, the elements of productivity are consistently categorized.

Because productivity measurements are only as valid as the primitive data used to calculate them, accuracy and consistency of the data used are essential for the results to be meaningful. The degree of granularity, which is the depth or level of detail at which data is collected, is important in determining the precision and consistency of the data.

Data of fine granularity is obtained when collecting data by staff-hour for each hour worked. Tracking the effort expended at this level reduces the potential for large errors. Estimates, such as those made at the end of a project by best guess approximation of effort, are an example of very coarse granularity. In this situation, large errors may be introduced into the productivity results.

This standard requires the use of the finest degree of granularity obtainable in collecting software productivity data. For the results of this standard to be most useful, the measuring process used shall be both accurate and detailed. Therefore, recording input effort as it is expended is the only acceptable method for this standard.

5. Output primitives

Three categories of output primitives are defined in this standard: source statements, function points, and documents. Source statements are a fundamental part of software, representing the encoded logic of the software product. Function points are a measure of the functional content of software. Production of documents is a significant component of the effort expended in software development, and measuring productivity for it is important. This standard requires reporting productivity results for source statements and documents. The use of function points is optional but recommended.

5.1 Source statement output primitives

This clause defines methods for counting software statements, and methods for categorizing the nature, origin, and disposition of the software.

Two methods are used to count software source statements: the logical source statement (LSS) method, which counts instructions, and the physical source statement (PSS) method, which counts lines of code. Logical source statement counting is the preferred method for this standard. Physical source statement counting is included in this standard because it is commonly found in published data.

Software is categorized by a set of attributes that describe it. Each source statement primitive is defined by three attribute qualifiers. The first of these is the type attribute that classifies each source statement as either executable, data declaration, compiler directive, or comment. The second is the origin attribute that classifies software as either developed or nondeveloped. Developed software is created new, or modified from existing software, for this product. Nondeveloped software is reused or deleted from the existing software. The last attribute is the usage attribute that identifies whether the software was delivered to a user of the final product, or produced only to support the development process and not delivered to the user.

5.1.1 Source statement counting

Source statements in different programming languages shall be counted separately. The programming language used shall be identified for each count along with its software module name. If two languages are used in the development, for example Pascal and C, the quantity of source statements of each language shall be counted and reported separately. Combining counts of different languages into a single count shall not be permitted.

This standard requires that only those source statements written by people be counted. Other counting practices, such as counting compiler derived instructions, program generator output, reuse library output, or machine instructions may be counted in addition to the people-created source statements provided these counts are kept separate and clearly identified. Counts of object code, executable code, and code that is mechanically generated from source code are not required by this standard. It may be most useful to count source statements at the level that the source is maintained. As a minimum, counting shall be done at product completion, when the product is ready for delivery. In all cases, what is counted shall be specifically stated along with the quantity of software measured.

Source statements that are expanded within a software module, for example, macro expansions, or supplied to the program, for example, by an *include* statement, shall be counted only once for all modules being measured. Source statements that invoke, call, or direct inclusion of other source statements in the module shall be counted each time they are used.

Source statements are counted in one of two ways, as logical source statements (LSS) or as physical source statements (PSS):

a) *Logical source statements*. The LSS counting method measures the number of software instructions. When a source statement is the count of instructions irrespective of its relationship to lines, the count shall be stated as being in logical source statements.

   The LSS method is used to count software statements independently of the physical format in which they appear. For example, if two instructions appear on a single line, the LSS count would be two. If one instruction spans two lines, the LSS count would be one.

   An LSS count shall be accompanied by a description of the procedure for counting the source statements. This description shall define the algorithm used to determine the beginning and end of each source statement, and how embedded source statements are counted.

b) *Physical source statements*. The PSS counting method measures the number of software lines of code. When the end-of-line character or a line of code method is used to count source statements, the count shall be stated as being in physical source statements.

   A PSS measurement is line counting where a typical line format is 80 characters. Any deviation from the typical line format shall be noted. A PSS may consist of one LSS, more than one LSS, or part of an LSS.

   Blank lines shall be excluded from PSS counts. If desired, blank lines may be counted separately as the number of blank lines.

5.1.2 Source statement attributes

This standard partitions source statements into three attributes: type, origin, and usage. The type attribute consists of executable, data declaration, compiler directive, and comment source statements. The origin attribute consists of developed and nondeveloped source statements. The usage attribute consists of delivered and nondelivered source statements.

Separate counts for each combination of attribute values shall be stated when reporting productivity results. For example, separate counts shall be stated for executable source statements that are developed delivered and for executable source statements that are developed nondelivered.

5.1.2.1 Type attribute

The type attribute classifies all LSS into the categories of executable, data declaration, compiler directive, and comment source statements. Each LSS shall be counted as one of the following:

a) *Executable source statements*. Source statements that direct the actions of the computer at run time shall be counted as executable source statements.
b) *Data declaration source statements*. Source statements that reserve or initialize memory at compilation time; that are directives to the compiler to constrain or check data; or that define the structure of the data that will be reserved, allocated, or created at run time shall be counted as data declaration source statements.
c) *Compiler directive source statements*. Source statements that define macros; are labels; direct the compiler to insert external source statements, for example, *include* statement; direct conditional compilation; or are not described by one of the other type attributes shall be identified as compiler directive source statements.
d) *Comment source statements*. Source statements that provide information to people reading the software source code and are ignored by the compiler shall be counted separately from other source statements and reported as comment source statements.

Physical source statement counting is a count of the number of nonblank lines. This count is expressed as the number of PSS. Physical source statements are further subdivided as follows:

a) *Noncomment lines*. Lines of software that contain either executable, data declaration, or compiler directive source statements.
b) *Comment lines*. Lines that contain only comment source statements.

All counts of LSS and PSS shall be kept by source module, and identified by module name and software language. The counts may be combined for all modules when expressing aggregate results for the software product, but the combined count shall be only of those modules of the same software source language.

5.1.2.2 Origin attribute

Each source statement is designated by a second attribute called the origin attribute. Each source statement can have one of two possible origins:

a) *Developed source statements*. Source statements added or modified for the specific product being measured shall be counted as new source statements. Added source statements are those that did not previously exist and were created specifically for this product. Modified source statements are those taken from another software product to which any changes were made to make the software suitable for this application.
b) *Nondeveloped source statements*. Source statements that were not developed new for this product are counted in two categories: deleted source statements and reused source statements. Deleted source statements shall be the count of all source statements that have been removed or modified from an earlier version of this software product. Reused source statements are unmodified source statements obtained for the product from an external source of software. The size (LSS or PSS) of this external source would be counted as original source statements and could be a previous version of this product, a reuse library, or acquired software. Reused source statements shall be counted as a part of the software module count. Modifying a source statement is considered to be the process of removing one existing source statement and then adding one or more new source statements. When a single source statement is modified, and the result is two or more source statements, all but one source statement shall be counted in added source statements, and one shall be counted as a modified source statement.

The following computations shall be used for calculating the counts of new, reused, and deleted source statements:

Number of New SS = Number of Added SS + Number of Modified SS

Number of Deleted SS = Number of Modified SS + Number of Removed SS

Number of Reused SS = Number of Original SS – Number of Modified SS – Number of Removed SS

The modified source statement count is identical in each relationship above in which it is used.

Pictorially, these relationships can be represented as in figure 1.

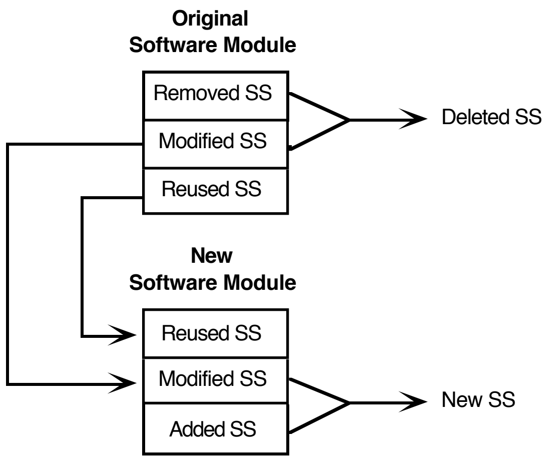


Figure 1—Source statements counts

5.1.2.3 Usage attribute

All source statements shall be divided into two categories based upon their usage:

a) *Delivered source statements*. All source statements incorporated into the final product delivered to the customer, shall be counted as delivered source statements. This includes all source statements that are used directly in the product's designed functions, and all source statements that support the product in its operating environment such as installation, diagnostic, and support utility software.
b) *Nondelivered source statements*. All software developed for the final product but not delivered to the customer shall be counted as nondelivered source statements. This includes all software not counted as delivered source statements that was produced to support development of the final product such as test software, tools, and aids to the developers.

5.2 Function point output primitive

The output primitive used to measure delivered software functionality is the function point. This standard uses the term *function point* to mean Albrecht's definition of function point (see [B1[2]]) or similar methods of measuring functionality.

Measuring function points is not required by this standard. However, those who do measure function points shall also accompany the measurements with source statement counts as defined in clause 5.1. This requirement is meant to enable a better understanding of the relationship between function point and source statement measures of productivity.

5.2.1 Function point counting

Function points are computed from an algorithm that uses various characteristics of the application's functionality expressed as a relationship of function types. Each function type is a variable that defines a characteristic of the application, and is set to a specific value for each application called the function value.

Productivity measurement given in function points shall be accompanied by the following:

a) The function point algorithm shall be stated explicitly. If the algorithm is proprietary, and cannot be stated, then the source and version of the algorithm shall be identified.
b) Each function type used in the function point algorithm shall be specified and defined. The function type definitions shall be complete (see 5.2.2).
c) A listing of the function values used for each function type shall be stated. This is the numeric quantity or factor supplied for each function type in the algorithm.

5.2.2 Function point attributes

The type attribute describes the function types. The origin and usage attributes are not generally used to describe function points since function points are computed only for delivered software.

Function types used in the function point algorithm shall be precisely defined. An example of how a function type definition might be stated is

> *External output type:* Any unique user data or control output type that leaves the boundary of the application being measured. An external output should be considered unique if it has a format different from other external outputs, or if the external (logical) design requires that processing logic be different from other external outputs with the same format. External outputs include reports and messages that leave the application directly for the user, and output files of reports and messages that leave the application boundary for other applications.

A definition is required for each function type used in the function point algorithm.

5.3 Document output primitives

This standard requires measurement of all documents that consume a nontrivial (as determined by the user of the standard) amount of project resources. Documents include hard copy, screen images, text, and graphics used to convey information to people. The effort associated with the preparation of each document measured shall be recorded separately.

[2]The numbers in brackets correspond to those of the bibliographical references in Annex C.

All documents that actively support the development or the usage of a software product shall be measured. Typical documents of this type are requirements statements, architectural and design specifications, data definitions, user manuals, reference manuals, tutorials, training guides, installation and maintenance manuals, and test plans.

Documents that are produced, but not preserved beyond the completion of the project should also be measured if they require a significant amount of effort to produce. These include such documents as proposals, schedules, budgets, project plans, and reports.

5.3.1 Document counting

The primitive unit of measure for a document is the page. This is a physical page for hard-copy documents, and a screen or page image for electronically displayed documents. The primary form of the output determines which measure is used.

The document measurement is augmented by reporting the page (or screen) size and the counts of three kinds of tokens: words, ideograms, and graphics. These auxiliary details help to distinguish between different document sizes and densities.

5.3.1.1 Document page count

The document page count is defined to be the total number of nonblank pages contained in a hard-copy document. For documents that are not printed, such as computer displays, a document screen count replaces the document page count.

The document page count is an integer value. Partially filled pages, such as at clause and chapter breaks, are counted as full pages. Blank pages are not counted.

5.3.1.2 Document page size

The page size for the document shall be specified. Both the width and the height dimensions of the document shall be measured in inches or equivalent units. This is the edge-to-edge dimensions, not the margin-to-margin dimensions.

For electronically displayed documents, the screen dimensions shall be specified. The screen width is measured by the average number of characters per line, and the screen height is measured by the number of lines per screen.

5.3.1.3 Document token count

The total number of tokens contained in each document are counted to characterize the document. Three kinds of token counts shall be made: words, ideograms, and graphics. The counting rules for each are as follows:

a) *Number of words*. When counting words in the text, these rules shall apply:
   — All simple words are counted as single words.
   — Contractions, such as *can't*, *won't*, *aren't*, are counted as single words.
   — Numeric values, such as *1234*, *12.34*, and *$1 234 567.89*, are counted as single words.
   — Acronyms, Roman numerals, abbreviations, and hyphenated words are counted as single words.
   — Punctuation marks are ignored.
b) *Number of ideograms*. These are symbols representing ideas rather than words, such as Kanji characters, or equations. When counting ideograms, each type of symbol is counted and reported separately, for example, the number of Kanji characters, and the number of equations.

c) *Number of graphics*. These are the number of graphs, tables, figures, charts, pictures, etc., that are included in the document. When counting graphics, the number of each type of graphic is reported separately, for example, the number of tables, the number of figures, the number of graphs, the number of pictures.

5.3.2 Document attributes

Document attributes define the type, origin, and usage of the documents measured.

5.3.2.1 Document type

Document type is reported with two values: the name of the document and its purpose. The name of the document is the name used by the development organization. The document purpose describes the intention of the document phrased in commonly used terminology.

5.3.2.2 Document origin

The amount of reuse of documents may be specified by the document's percentage reused. This value is computed by dividing the number of tokens in the document that are not modified or added by the total number of tokens in the document. The result is expressed as a percentage of the type of tokens measured. The type of tokens used for calculating the percentage shall be stated. For example, if a document contains 8000 word tokens from the original document that were not modified or added, and the final document size is 10 000 word tokens, then the document would be reported to have a value for the document percentage reused of 80% word tokens (8000/10 000).

5.3.2.3 Document usage

Each document shall be identified as to its usage, delivered or nondelivered. The document usage is delivered if it is provided to the user or to the user's facility. A nondelivered document is one that is not provided to the user.

6. Input primitive

In the productivity relationship for this standard, input represents the effort applied to develop the software product. The principal measure for effort in this standard is labor expressed in units of staff-hour.

6.1 Staff-hour input primitive

Effort shall be measured only in staff-hour units. A staff-hour is an hour of time expended by a member of the staff. The staff-hours collected shall include those directly expended in creating a specific output product for which productivity is calculated.

The accuracy of productivity results depends on how precisely the effort expended to produce an output product is recorded. Recording effort as it is expended is required by this standard.

The staff-hour measure shall include all effort expended on the product, including all overtime hours, both compensated and uncompensated.

Staff-hours are not counted in productivity calculations if they do not directly contribute to the software product. Any time away from the job, such as vacations, holidays, jury duty, and sick leave, are not counted. Also, time expended on something other than the specific product being measured, or time spent on other products or projects is not counted.

Staff-hours may be expressed as larger units for convenience. To express effort in larger units, such as staff-day, staff-month, or staff-year, a conversion factor shall be specified to translate the larger unit into staff-hours. For example, stating results in staff-months, the user would accompany it with a statement similar to "one staff-month = 152 staff-hours for this data."

6.2 Staff-hour attribute

The nature of the effort expended by personnel shall be measured in the following categories:

a) *Direct staff-hour.* The effort expended by those members of the development staff that directly contribute to defining or creating the software product is measured as direct staff-hours. This category should include, but not be limited to, first level supervisors and members of the staff that perform analysis, design, programming, testing, and documentation tasks.
   1) *Direct delivered staff-hour.* All direct staff-hours resulting in the production of output primitives (namely, source statements, function points, or documents) that are delivered to the customer shall be counted as direct delivered staff-hours.
   2) *Direct nondelivered staff-hour.* All direct staff-hours resulting in the production of output primitives (namely, source statements, function points, or documents) that are not delivered to the customer shall be counted as direct nondelivered staff-hours.
b) *Support staff-hour.* The effort expended by those members of the staff who do not directly define or create the software product, but act to assist those who do, shall be collected and recorded as support staff-hours. This category should include, but not be limited to, members of the staff who perform clerical functions or support activities, such as development computer operators or project coordinators. This category also includes those who contribute to the production process but whose efforts are too distributed to allocate their effort directly to a specific product, such as configuration managers and quality assurance personnel.

6.3 Activities

This standard does not specify a standard life cycle, nor does it imply the existence of a standard software life cycle. Users of this software standard shall specify activities of their own life cycle that directly contribute to producing specific output primitives in order to compute the productivity associated with those outputs (see IEEE Std 1074-1991 for life cycle activities). Staff-hours shall be collected and reported for each output primitive by staff-hour attribute.

7. Relationships

This clause describes several types of relationships, using the primitives defined in clauses 5 and 6. These relationships are presented as examples of the way that output and input primitives may be combined. It is left to the users to decide which combinations of primitives are most useful to them.

7.1 Productivity ratios

In this standard, productivity is defined as the ratio of the output product to the input effort that produced it. For a specific product output, such as the number of source statements or the number of document pages, productivity is computed by dividing that product quantity by the effort expended on it.

Productivity is computed using the ratio of output to input for each measured output primitive for which related input effort data is collected. The productivity for product *a*, which has the quantity of output primitive O_a and which required effort E_a to create, is defined by the relationship

$$\text{Productivity}_a = \frac{O_a}{E_a}$$

The ratio's units of measure are those from the dimensions of the primitives used, for example, source statements per staff-hour, or document page count per staff-hour.

All statements of productivity shall be accompanied by information about the characteristics (see clause 8) and about the activities included in the measurement (see IEEE Std 1074-1991).

The usefulness of productivity values depends on the accuracy of the data used to compute them. Here accuracy refers to the level of detail and precision of the data collection process; estimates are not acceptable. Only recording the input effort as it is worked and counting the output produced are acceptable methods for measuring the primitives.

Only the product quantity and the effort expended to build that product should be used in calculating productivity. Any degree of ambiguity introduced in measuring the output product or capturing the effort diminishes the usefulness of the productivity value. For example, consider computing the productivity of a software module when the effort is not specifically recorded for it. If the effort for writing source statements was not recorded separately, but included in the effort of creating documentation for it, the resulting productivity would be the output of source statements divided by the effort for source statements and documentation. This result would be difficult to interpret because the productivity would not be for source statements alone.

7.1.1 Incremental productivity ratios

Productivity shall be computed at the completion of an output product, that is, when the product is delivered to the user. At the point of delivery, the productivity is

$$\text{Final Productivity} = \frac{O_{\text{Final}}}{E_{\text{Final}}}$$

where O_{Final} is the total quantity of the output product being measured, and E_{Final} is the total effort expended in creating it.

This relationship might also be used during the development process prior to delivery to ascertain accumulated productivity results. In this case the output measured is the quantity of the product at that point in the process, and the input is the effort expended in producing that incremental quantity of output.

During development of a product, it may be useful to measure productivity at intervals to assess trends as it progresses toward final productivity. This periodic sampling of productivity is called the incremental productivity and computed by

$$\text{Incremental Productivity} = \frac{O_{tn} - O_{tn-1}}{E_{tn} - E_{tn-1}}$$

where O_{tn} is the output production and E_{tn} is the direct effort for that output expended through time t_n, and O_{tn-1} and E_{tn-1} are the output production and direct effort through some prior time t_{n-1}.

If the incremental productivity is computed at intervals throughout the development process, it may show productivity fluctuations. It is useful to chart the progress of productivity in order to show trends, or to estimate final productivity based on trend patterns from past projects.

7.1.2 Source statement productivity ratios

The source statement output primitives that may be useful in productivity calculations are:

Delivered New SS

Delivered Reused SS

Nondelivered New SS

Nondelivered Reused SS

Nondelivered Deleted SS

where SS stands for logical source statements or physical source statements.

The input primitives used in productivity relationships are:

Direct Delivered Staff-Hour

Direct Nondelivered Staff-Hour

Support Staff-Hour

Ratios of output primitives to input primitives will show various types of productivity results. For example, the productivity for direct delivered source statements may be computed for the output primitives:

Delivered New SS

Delivered Reused SS

by dividing each by the input primitive for

Direct Delivered Staff-Hours

The productivity for direct nondelivered source statements is computed by taking each of the output primitives:

Nondelivered New SS

Nondelivered Reused SS

Nondelivered Deleted SS

and dividing by the input primitive:

Direct Nondelivered Staff-Hours

The differences between software languages are sufficiently great to make cross-language comparisons of productivity difficult. This standard provides a foundation for collecting accurate productivity data. This may make cross-language comparisons possible in the future.

This standard supports only comparing source statement productivity of identical source languages. When attempting to compare the source statement productivity of two software developments, they shall be in the same software programming languages. This means that the productivity computed for a software product development in a third-generation language shall not be directly compared with the productivity for a software product developed in an assembler-level language, nor should it be compared with a different third-generation language.

7.1.3 Function point productivity ratios

The function point output primitive is the number of function points.

The input primitives used in productivity relationships are:

Direct Delivered Staff-Hour

Direct Nondelivered Staff-Hour

Support Staff-Hour

Dividing the number of function points by each of the above input primitives provides different productivity ratios.

7.1.4 Documentation productivity ratios

Documentation is another form of software output, and document outputs are treated similarly to source statement outputs.

The document output primitives are:

Document Page Count or Screen Count

Number of Words

Number of Ideograms

Number of Graphics

The input primitives used for document productivity are:

Direct Delivered Staff-Hour

Direct Nondelivered Staff-Hour

Support Staff-Hour

All of the productivity ratios described for source statements apply analogously for documents.

7.2 Output-to-output ratios

The following clauses describe some output-to-output ratios. Other combinations of output-to-output ratios will provide additional insights into the process being measured.

7.2.1 Source statement output-to-output ratios

Consider the matrix of the source statement origin to usage attributes in table 1.

Table 1—Source statement origin to usage attribute matrix

		USAGE	
		Delivered	Nondelivered
ORIGIN	Developed	New SS (1)	New SS (2)
	Nondeveloped	Reused SS (3)	Reused SS (4)
		Deleted SS* (5)	Deleted SS (6)

*Measurement data for this variable is not collected, since deleted software is not delivered to the user.

Output source statement primitives in the cells of the preceding table combine to form numerous relationships. Some of these are:

The total number of developed source statements:

Developed SS = (1) + (2)

The total number of delivered source statements:

Delivered SS = (1) + (3)

All source statements contributing to the total product:

Total SS = (1) + (2) + (3) + (4)

The total number of source statements not delivered to the user:

Nondelivered SS = (2) + (4) + (6)

The total number of reused source statements:

Reused SS = (3) + (4)

The total number of source statements deleted from the product prior to delivery to the user:

Deleted SS = (6)

The proportion of delivered source statements to the number of source statements in the total output:

$$\frac{\text{Delivered SS}}{\text{Total SS}} = \frac{(1) + (3)}{(1) + (2) + (3) + (4)}$$

The proportion of reuse in the total product:

$$\frac{\text{Reused SS}}{\text{Total SS}} = \frac{(3) + (4)}{(1) + (2) + (3) + (4)}$$

The proportion of delivered source statements that are new:

$$\frac{\text{New}_{\text{Delv}}\ \text{SS}}{\text{Delivered SS}} = \frac{(1)}{(1) + (3)}$$

The proportion of delivered source statements that are reused:

$$\frac{\text{Reused}_{\text{Delv}}\ \text{SS}}{\text{Delivered SS}} = \frac{(3)}{(1) + (3)}$$

The proportion of nondelivered source statements that are reused:

$$\frac{\text{Reused}_{\text{Nondelv}}\ \text{SS}}{\text{Nondelivered SS}} = \frac{(4)}{(2) + (4) + (6)}$$

Ratios of the type attribute values may prove to be useful. Consider the type attribute values for logical source statements:

Executable LSS	(7)
Data Declaration LSS	(8)
Compiler Directive LSS	(9)
Comment LSS	(10)

A ratio of interest might be the executable LSS to the total LSS for a module:

$$\frac{\text{Executable SS}}{\text{Module's Total LSS}} = \frac{(7)}{(7) + (8) + (9) + (10)}$$

Consider similar ratios for the values of the PSS type attribute:

Noncomment PSS	(11)
Comment PSS	(12)

Finally, consider the ratio of comments to the total number of PSS:

$$\frac{\text{Comment PSS}}{\text{Total PSS}} = \frac{(12)}{(11) + 12}$$

7.2.2 Function point output-to-output ratios

Output-to-output ratios are meaningless for the cases where the function point count is a single number. In other cases, ratios similar to those above can be constructed.

7.2.3 Documentation output-to-output ratios

Three document content ratios are computed to describe the density of a document. These ratios help to distinguish high density documents from low density ones:

Number of Words per Page

Number of Ideograms per Page

Number of Graphics per Page

Document usage ratios are equivalent to the source statement usage ratios. Measurement of delivered documents versus nondelivered documents is at the document level; either the document is delivered or it is nondelivered. Measurement of developed versus nondeveloped documents shall be made at the token level; the page level is inadequate due to page break realignments and changes made to tokens.

Document products are developed in tokens, but often are measured in pages or screens. The origin to usage matrix for document pages is shown in table 2. A similar matrix could be made for screens or for each of the individual tokens, namely, words, ideograms, or graphics.

Table 2—Document origin to usage attribute matrix

		USAGE	
		Delivered	Nondelivered
ORIGIN	Developed	New Pages (1)	New Pages (2)
	Nondeveloped	Reused Pages (3)	Reused Pages (4)
		Deleted Pages* (5)	Deleted Pages (6)

*Measurement data for this variable is not collected, since deleted pages are not delivered to the user.

The document output-to-output terminology and ratio definitions are applied at the token level, and are similar to the terminology and ratio definitions described for source statement output-to-output ratios.

7.2.4 Mixed output primitive ratios

Ratios comparing output primitives of different types may provide insight into the software development process. Two possible combinations are shown here for consideration:

$$\frac{\text{Function Points}}{\text{Total Source Statements}} \quad \text{or} \quad \frac{\text{Document Pages}}{\text{Total Source Statements}}$$

7.3 Input-to-input ratios

Ratios of input primitives to input effort primitives provide insights about the effort expended. The input primitives are

Direct Delivered Staff-Hours	(A)
Direct Nondelivered Staff-Hours	(B)
Support Staff-Hours	(C)

Some meaningful combinations with these primitives are

Total Effort = (A) + (B) + (C) and Total Direct Effort = (A) + (B)

The ratio of Direct Effort to Total Effort:

$$\frac{\text{Total Direct Effort}}{\text{Total Effort}} = \frac{(A) + (B)}{(A) + (B) + (C)}$$

The ratio of Support Effort to Total Direct Effort:

$$\frac{\text{Support Effort}}{\text{Total Direct Effort}} = \frac{(C)}{(A) + (B)}$$

The ratio of Direct Delivered to Direct Nondelivered Effort:

$$\frac{\text{Direct Delivered Effort}}{\text{Direct Nondelivered Effort}} = \frac{(A)}{(B)}$$

Many other meaningful combinations are possible. Users of this standard may wish to explore them.

8. Characteristics

Within the context of this document, characteristics are defined as those inherent factors of software development that may have a significant impact on productivity. The purpose of recording measures of characteristics is to ensure consistent comparisons between projects developing software products.

The characteristics are divided into three categories: project characteristics, management characteristics, and product characteristics. Project characteristics describe the people and processes involved in the development effort. Management characteristics describe how a project is managed. Product characteristics reflect the nature of the product itself.

Whenever data is collected to assess software productivity, the characteristics shall be recorded. Comparing productivity data between projects is applicable only if the characteristics are similar. This standard does not attempt to precisely quantify characteristics, but it does require that all characteristics be documented and reported along with productivity data.

Most of the required characteristics data shall be recorded in the early stages of the project. Some data, however, will not be available until the later stages. Characteristics shall be monitored throughout the project and recorded in the characteristics clause as new data is available.

The following is the minimum set of characteristics that shall be recorded.

8.1 Project characteristics

Project characteristics are factors involving the people and the working environment of the development effort. They represent factors that the developer may be able to control or alter. Project characteristic information is recorded in one of two main categories: personnel or software development environment (SDE).

8.1.1 Personnel

The personnel category includes most of the human elements that are involved in a software development project, such as education, experience, expert experience, training, size, and turnover. These personnel characteristics are explained below, and when possible, a metric is included for each characteristic.

8.1.1.1 Education

Education refers to the highest level of education beyond high school.

Level of education:

— No Degree
— Associate’s Degree
— Bachelor’s Degree
— Master’s Degree
— Doctorate Degree

Classify each member of the project team according to the above list. Record this data as an educational profile expressed as a percentage of total team (for example, 10% Associate’s, 50% Bachelor’s, 30% Master’s, 10% Doctorate degrees).

8.1.1.2 Experience

Experience refers to the number of years of relevant experience, and encompasses three aspects:

a) Years of work in software engineering
b) Years of work with specific software applications and technologies relevant to this project (for example, 4GL, graphics, real-time)
c) Years of experience in the organization

Record years of relevant experience as a profile in the three categories for the highest, lowest, and median years (for example, Experience in current job: Highest = 8 yrs, Lowest = <1 yr, Median = 2.5 yrs).

8.1.1.3 Expert assistance

Expert assistance refers to a person, or group of persons, external to the project team who has knowledge and experience in the application being developed by the project team.

Indicate if expert assistance was required on this project. Record expert assistance required as either yes or no.

Indicate if expert assistance was available as needed for this project. Record expert assistance availability as either yes or no.

8.1.1.4 Training

Training refers to all formal professional training. This includes external training courses, such as seminars, and in-house training courses, such as training on company coding standards, software development environment, testing techniques, etc.

Record the average number of days each project employee spends in training per year (for example, an employee receives an average of 6.5 days of training per year).

8.1.1.5 Size

Size refers to the number of direct and support staff involved in the project. Record the number of people on the project at peak and average staff levels.

8.1.1.6 Turnover

Turnover refers to a shift in personnel during a project. Record if turnover was a disruptive factor as either yes or no.

8.1.2 Software development environment

The software development environment (SDE) is the combination of tools, techniques, and administration used during the development. For recording this part of the project characteristics, below is a checklist of the items to be included:

a) On-line documentation with word processing
b) Configuration management techniques and tools
c) Project libraries
d) Formal design techniques
e) Coding standards
f) Format and content standards for documentation
g) Software cost estimation tools
h) Formal project management methods (for example, PERT charts, work breakdown structures, critical path analysis, etc.)
i) Reuse libraries
j) Walkthroughs, code inspections, design reviews, etc.
k) Automated test tools
l) Debugging tools
m) Defect tracking and resolution systems
n) Requirements traceability
o) Other

Record all letters that are used by this project.

8.2 Management characteristics

This clause provides some insight into how the project was managed. The data for this clause should be recorded after the project is completed.

8.2.1 User participation

User participation refers to the level of participation by the user or their representative on the project.

— *High* is user participation in all reviews, as well as commenting on all documentation.
— *Low* is user participation limited to just the start and end of the project.
— *Medium* is all user participation between the levels of *High* and *Low*.
— *N/A* is when user participation is not applicable.

Record the appropriate level of user participation.

8.2.2 Stability of product requirements

Stability of product requirements characterizes the extent that the requirements remained constant throughout development. Record whether instability of product requirements was a disruptive factor as either yes or no.

8.2.3 Constraining factors

Specify the factors that constrained the project. The following is a list of possible constraining factors:

— Fixed cost
— Fixed staff size
— Fixed functionality
— Fixed quality and/or reliability
— Fixed schedule
— Limited accessibility to development system
— Limited accessibility to target system

Record these and any other limiting factors as a checklist (for example, cost, schedule, access to target system).

8.3 Product characteristics

Product characteristics are those factors imposed upon the product. The specific product characteristics that shall be recorded are defined below.

8.3.1 Criticality

The following are product critical characteristics that directly impact productivity.

8.3.1.1 Timing critical

Certain products must work in environments where the real-time behavior, user response, or throughput is critical. Record timing critical as either yes or no.

8.3.1.2 Memory critical

Certain products must fit into a limited amount of memory. Record memory critical as either yes or no.

8.3.1.3 Quality/reliability critical

Certain products must meet very stringent quality or reliability criteria. Record quality/reliability critical as either yes or no.

8.3.2 Degree of innovation

This characteristic represents the technological risk of the project.

0 = Never done before

1 = Previously done by others

2 = Previously done by us

Record the degree of innovation as either 0, 1, or 2.

8.3.3 Complexity

This standard requires a rating for three types of complexities: programming, data, and organizational.

8.3.3.1 Programming complexity

Programming complexity addresses the program control flow of a software product.

— *Low* is straight line code, simple loops and branches, simple calculations.
— *Medium* is simple nesting, simple intermodule communications, moderately difficult calculations.
— *High* is re-entrant/recursive code, data structure control, complex calculations requiring accurate results.

Record programming complexity. Other formal complexity measures may be recorded as well. In this case, the complexity measurement methodology used shall be cited.

8.3.3.2 Data complexity

Data complexity refers to the arrangement, access, and retrieval of stored data.

— *Low* are simple arrays or files in main memory, few inputs/outputs, no restructuring of data.
— *Medium* are multiple inputs/outputs, data typing, restructuring of data, access to other storage media (for example, tape, hard disk, other machines).
— *High* are highly coupled and relational data structures, optimized search algorithms, background data consistency checks.

Record data complexity.

8.3.3.3 Organizational complexity

Organizational complexity refers to the difficulty in coordinating and communicating with all parties on the project team.

SP = Single Person

SD = Single Department/Division

MD = Multiple Departments/Divisions

MS = Multiple Sites

SC = Subcontracted development to third party

Record all levels of organizational complexity that apply (for example, MD, MS).

8.3.4 Development concurrency

Some products require concurrent, that is, parallel development of software (S), hardware (H), and firmware (F).

H-S: hardware and software concurrent

H-F: hardware and firmware concurrent

S-F: software and firmware concurrent

S-S: software and software concurrent

H-S-F: hardware, software, and firmware concurrent

Record which areas were being developed concurrently.

8.3.5 Product description

Provide a brief description of the software application, that is, a brief description of what the product does.

Annexes

(These informative annexes are not a part of IEEE Std 1045-1992, IEEE Standard for Software Productivity Metrics, but are included for information only.)

Annex A Sample metrics data collection summary list

(informative)

The items listed here are required by the standard. Items inside brackets “[...]” are optional or may be required under special conditions. Numbers in parentheses “(...)” are the clause numbers of this standard that describe that measurement.

A.1 Descriptive fields

This information identifies the source of the measurements.

— Product Name
— Product Release and Version
— Development Organization
— Current Date
— Name of Data Collector

A.2 Software module information (5.1.1)

— Software Module Name
— Software Language

A.3 Type attribute primitives (5.1.2.1)

— Logical Source Statement (LSS) Count:
  • Number of Executable LSS
  • Number of Data Declarations
  • Number of Compiler Directives
  • Number of Comments

— Physical Source Statement (PSS) Count:
  • Number of Noncomment Lines
  • Number of Comment Lines
  • Number of Characters per Line (if different from 80 characters per line)
  • [Number of Blank Lines]

A.4 Origin attribute source statement primitives (5.1.2)

— Number of Developed Source Statements
— Number of Nondeveloped Source Statements
— Source Statement Counts:
  • Number of Original SS
  • Number of Added SS
  • Number of Modified SS
  • Number of Removed SS

A.5 Usage attribute source statement primitives (5.1.2.3)

— Number of Delivered Source Statements
— Number of Nondelivered Source Statements

A.6 Function point primitives (5.2.1)

— Number of Function Points
— Function Point Algorithm
— Function Types Description
— Function Type Values for Each Function Type

A.7 Documentation primitives (5.3)

— Document Name
— Document Purpose
— Document Usage (delivered or nondelivered)
— Document Medium (hard copy or screen)
— For Hard-Copy Documents:
  • Document Page Count (blank pages not counted)
  • Page Size (edge-to-edge in inches)
— For Electronically Displayed Documents:
  • Document Screen Count
  • Average Number of Screen Characters per Line
  • Lines per Screen
— Number of Words
— Number of Ideograms
— Number of Graphics (list each type separately)
— [Document Percentage Reused (name the token measured)]

A.8 Input primitives by activity category (6.0)

— Activity Name
— Direct Delivered Staff-Hours
— Direct Nondelivered Staff-Hours
— Support Staff-Hours

Annex B
Characteristics data collection form

(informative)

Project Characteristics (8.1)

Personnel

Education	Percent of Project Team Members
No Degree:	_______%
Associate's Degree:	_______%
Bachelor's Degree:	_______%
Master's Degree:	_______%
Doctorate Degree:	_______%

Experience

1) Years in software engineering:	Highest____ Lowest____ Median____
2) Years in specific s/w technology:	Highest____ Lowest____ Median____
3) Years in organization:	Highest____ Lowest____ Median____

Expert Assistance	Required:	________ YES/NO
	Available:	________ YES/NO
Training	Average:	________ days per year
Size	Peak:	________ number of people
	Average:	________ number of people
Turnover	Disruptive:	________ YES/NO

Software Development Environment: _________, _________, ________, _________

Management Characteristics (8.2)

User Participation		________ H/M/L/NA
Product Requirements Stability	Disruptive:	________ YES/NO
Constraining Factors		_________, _________, _________, _________

Product Characteristics (8.3)

Criticality

Timing Critical:	________ YES/NO
Memory Critical:	________ YES/NO
Quality/Reliability Critical:	________ YES/NO

Degree of Innovation: ________ 0/1/2

Complexity

Programming:	________ H/M/L
Data:	________ H/M/L
Organizational:	____, ____, ____ SP/SD/MD/MS/SC (Record all that apply.)

Development Concurrency: ________ H-S/H-F/S-F/S-S/H-S-F

Product Description:____

Annex C Bibliography

(informative)

The following references were used extensively in the development of this standard. Users of this standard may wish to become familiar with these and other software productivity literature to further their understanding of the subject.

[B1] Albrecht, Allan J., and Gaffney, John E., 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, Nov. 1983.

[B2] Basili, Victor R., and Weiss, David M., "A methodology for collecting valid software engineering data." *IEEE Transactions on Software Engineering,* vol. SE-10, no. 6, pp. 728–738, Nov. 1984.

[B3] Boehm, Barry W., *Software Engineering Economics,* Prentice Hall, Inc., Englewood Cliffs, NJ, 1981.

[B4] Conte, S. D., Dunsmore, H. E., Shen, V. Y., *Software Engineering Metrics and Models,* Benjamin/Cummings Publishing, Menlo Park, CA, 1986.

[B5] Grady, Robert B., Caswell, Deborah L., *Software Metrics: Establishing a Company-Wide Program,* Prentice Hall, Inc., Englewood Cliffs, NJ, 1987.

[B6] Jones, T. Capers, *Programming Productivity,* McGraw–Hill Book Company, New York, 1986.

[B7] Putnam, Lawrence H., "A general empirical solution to the macro software sizing and estimation problem." *IEEE Transactions on Software Engineering,* vol. SE-4, no. 4, pp. 345–361, Jul. 1978.

Annex D Software counting relationships

(informative)

Presented here is an example of several relationships among the various categories of source statements included in a software product or application system. Rules relating the counts of each of these categories are also provided.

A new software system may consist of code obtained from an existing one and of new code created specifically for it. The earlier system may be a member of the same family as the new one or may be a member of a different family. Formulas are provided here that relate the source statement counts for the several categories of statements involved in creating a new software system. The formulas and relationships presented apply to both logical and physical statements, with or without comments. However, in any given instance, they are to be applied to only one category of statement (for example, physical statements, without comments). The logical source statement (LSS) count measures the number of software instructions independent of their physical format. It is imperative that LSS counts be accompanied by a description of the way in which they were counted. The physical source statement (PSS) count measures the quantity of software in lines of code.

A system consists of two categories of code: *new* and *reused*. It is expected that the development labor costs for a new application system are to be assigned corresponding to these categories of code. The definitions of *new* and *reused* source statements are as follows:

new source statements: The number of LSS or PSS that were newly created or modified for the application or system.

reused source statements: The number of LSS or PSS incorporated unmodified into the application system. Code that is ported is considered to be reused under the definition here.

When a new system is created in part from an earlier one, some of the statements from the earlier one are deleted. Figure D1 is a Venn diagram that depicts the relationships between the three categories of statements, new, reused, and deleted.

The development of a new software system using some source statements from an existing one employs four types of operations: removal, addition, modification, and reuse of some of the original system's code. The counts of each of the four types of source statements correspond to those operations; removed (re), added (a), modified (m), and reused original (o); are elements of the counts for the more general categories, new (n), reused (r), and deleted (d), related as shown in figure D1.

a) new = added + modified or, $n = a + m$
b) deleted = modified + removed or, $d = m + re$
c) reused = original – deleted or, $r = o - m - re$

Note that *removed* means physically removed. If a source statement is changed in any way, it is called *modified*. Hence, the count of *new* statements is the sum of the *added* and the *modified* counts. Figure D2 is a Venn diagram that depicts the relationships between the sets of source statements of the different categories cited above. It shows the components of the new, reused, and deleted categories depicted in figure D1.

The formulas given above are for the special case of modifying one system to produce another. Note that the generalization works well, that is, incorporating code from several *original systems* into a *new* one. However, it is necessary to carefully define the quantities appropriately when doing so.

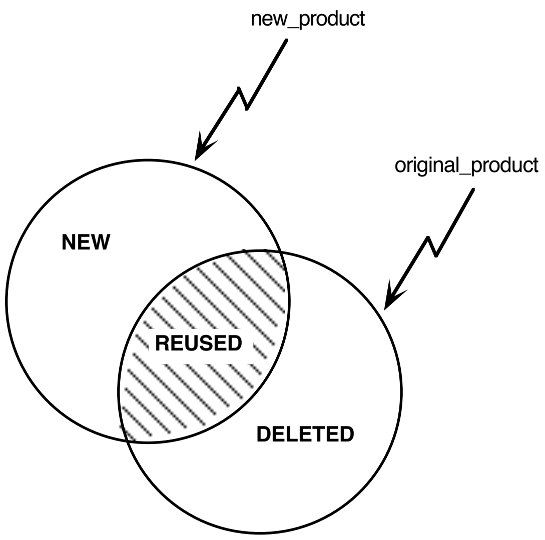
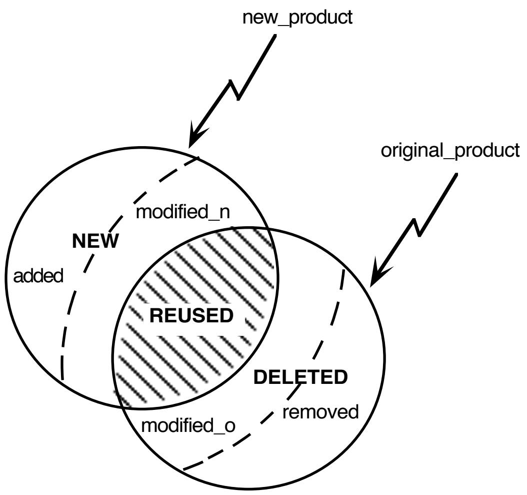


Figure D1—Statement relations



$$\text{new\_file} = \text{new} \cup \text{reused}$$
$$\text{old\_file} = \text{reused} \cup \text{deleted}$$
$$\text{diff\_file} = \text{new} \cup \text{deleted}$$
$$\text{new} = \text{added} \cup \text{modified\_n}$$
$$\text{deleted} = \text{removed} \cup \text{modified\_o}$$

NOTE—SS_count(modified_n) = SS_count(modified_o); that is both of these sets are the same size.

Figure D2—Detailed statement relations

Suppose there are three files of source corresponding to the original system (old_file), the new system (new_file), and the difference between the two (diff_file). The (diff_file) is the set of elements not found in both the (new_file) and the (old_file). The Venn diagram in figure D2 shows them to be the *new* and *deleted* elements. The counts indicated below show this. The counts of the reused source (r), the new source (n), and the deleted source (d), as defined above, can be obtained very simply from the counts of the three files. Let c() mean *count of.* Then the following relationships exist:

a) c(old_file) = reused + deleted = r + d.
   By the formulas given above,
   c(old_file) = (o – m – re) + (m + re) = o.

b) c(diff_file) = new + deleted = n + d.
   By the formulas given above,
   c(diff_file) = (a + m) + (m + re) = a + 2m + re

c) c(new_file) = reused + new = r + n.
   By the formulas given above,
   c(new_file) = (o – m – re) + (a + m) = a + o – re.

Observe that the diff_file count, c(diff_file), includes a factor 2m, twice the number of the modified source statements. The reason for this can be understood by considering the Venn diagram in figure D2. There are actually two sets of *modified* statements, both of size m. The first set is in the original system. The statements in that set are modified or transformed into a somewhat different set of statements in the new system. Note that the original or *old* system may have some statements in it that will be modified or changed and become *new* statements in the new system.

The counts for reused (r), new (n), and deleted (d) statements, as defined above, can be obtained from the counts for the old_file, the diff_file, and the new_file, as now shown. Verifications of the correctness of the formulas are also provided. The formulas are as follows:

a) ((c(old_file) + c(new_file)) – c(diff_file))/2 = r
   Verification; ((r + d + r + n) – (n + d))/2 = r,
   or, ((o + (a + o – re)) – (a + 2m + re))/2 = (2o – 2m – 2re)/2 = o – m – re = r,
      by definition.

b) c(new_file) – r = n
   Verification; (r + n) – r = n
   or, (a + o – re) – (o – m – re) = a + m = n,
      by definition.

c) c(old_file) – r = d
   Verification; (r + d) – r = d
   or, (o – (o – m – re)) = m + re = d,
      by definition.

An example is now provided for applying of the counting rules presented above. In it, a new module is created using some statements obtained from another module developed earlier. The counts for the various categories of statements involved are:

Statement Category	Statement Count
Original module size (o):	100
Statements modified (m):	7

Statements added (a):	3
Statements removed (re):	2

Therefore:

Statements reused = r = o – m – re = 100 – 7 – 2 = 91

Statements new = n = a + m = 3 + 7 = 10

Statements deleted = d = m + re = 7 + 2 = 9.

Consequently, the total product size, which is the sum of the new and the reused statements, is

10 + 91 = 101

The counts of the new (n), reused (r), and deleted statements (d), can be determined very simply using the relationships presented by the three equations in the text immediately preceding this example. Suppose the statement counts for the old_file, the new_file, and the diff_file were as follows:

a) c(old_file) = 100

b) c(new_file) = 101

c) c(diff_file) = 19

Then the desired counts can be determined as follows:

a) count of reused,
r = (c(old_file) + c(new_file) – c(diff_file))/2 = (100 + 101 – 19)/2 = 91.

b) count of new,
n = c(new_file) – r = 101 – 91 = 10.

c) count of deleted,
d = c(old_file) – r = 100 – 91 = 9.

These values are the same as those derived earlier in the example from the counts for modified, added, original, and removed, demonstrating the correctness of the calculations used to obtain them. The example illustrates the simplicity of calculating the number of new, reused, and deleted statements from the counts of the statements in the three files.