

IEEE Standard for a Software Quality Metrics Methodology

Sponsor

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

Approved December 13, 1992

IEEE Standards Board

Approved June 9, 1993

American National Standards Institute

Abstract: A methodology for establishing quality requirements and identifying, implementing, analyzing, and validating the process and product of software quality metrics is defined. The methodology spans the entire software life cycle. Although this standard includes examples of metrics, this standard does not prescribe specific metrics.

Keywords: direct metric, factor, metrics framework, software quality metric, subfactor

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

Copyright © 1993 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1993. Printed in the United States of America

ISBN 1-55937-277-X

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

IEEE Standards documents are adopted by the Institute of Electrical and Electronics Engineers without regard to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Introduction

(This introduction is not a part of IEEE Std 1061-1992, IEEE Standard for a Software Quality Metrics Methodology.)

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

Norman Schneidewind, *Chair*

Chris Baldwin and Elizabeth Banks, *Editors*

George Klammer, *Vice Chair*

Ron Braun and Craig Fuget, *Secretaries*

Martha Amis
James Anderson
Nelson Andrews
John Bowen
Fletcher Buckley
S.C. Chang
David Classick
Taz Daughtrey
Raymond Day
Deborah De Toma
Rose Der
James Dinkey
Ruth Euler
Michael Evangelist
David Favor

Thomas Kurihara
Hal Larson
Violet Foldes
Al Freund
Oscar Garcia
John Girard
Randy Greene
Warren Harrison
Clark Hay
Dan Hocking
Meinhard Hoffmann
Robert Holibaugh
John Horch
Bud Jones
William Junk
Maria Kaufmann

Philip Marriott
Denis Meredith
Mary Mikhail
Celia Modell
Jai Navlakha
Wilma Osborne
Art Price
Horst Richter
Carl Seddio
Raghu Singh
Karen Snow
William Turner
John Walker
Gene Walters
Paul Wolfgang



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

Wolf Arfvidson	Kirby Fortenberry	Ivano Mazza
Richard L. Aurbach	John Franklin	John P. McArdle
Motoei Azuma	David Gelperin	Celia H. Modell
Bruce M. Bakken	Yair Gershkovitch	Celia H. Modell
Bakul Banerjee	Shirley Gloss-Soler	Gerry Neidhart
David Barber	John Garth Glynn	Dennis E. Nickle
Boris Beizer	Victor M. Guarnera	Paul G. Petersen
Leo Beltracchi	David A. Gustafson	Shari Lawrence Pfleeger
Mordechai Ben-Menachem	George B. Hawthorne	John G. Phippen
H.R. Berlack	Mark Heinrich	Meir Razy
William J. Boll	Charles P. Hollocker	Frances A. Ruhlman
Kathleen L. Briggs	David Johnson III	Julio Sanz
Bruce Brocka	S. Jones	Stephen R. Schach
A. Winsor Brown	Richard Karcich	Hans Schaefer
William Bryan	S. Kasad	Norman Schneidewind
Fletcher Buckley	Peter Klopfenstein	Roger W. Scholten
Margaret K. Butler	Robert Kosinski	Gregory D. Schumacher
Jung K. Chung	Joseph Krupinski	Carl S. Seddio
Won L. Chung	Thomas M. Kurihara	Robert W. Shillato
Antonio Cicu	Lak Ming Lam	David M. Siefert
Francois Coallier	Rebecca Ann Lamb	Al Sorkowitz
Christopher Cooke	Renee Lamb	Richard Staunton
Stuart Corcoran	John B. Lane	Robert Thibodeau
Stewart Crawford	F.C. Lim	Leonard L. Tripp
Patricia W. Daggett	Bertil Lindberg	Mark-Rene Uchida
Taz Daughtrey	Ben Livson	Margaret Updike
Claudia Dencker	Dieter Look	Udo Voges
Peter A. Denny	J. Maayan	Dolores Wallace
Einar Dragstedt	Harold Mains	John W. Walz
Peter Dyson	Henry A. Malec	Paul A. Willis
Mary L. Eads	David M. Marks	Andrew Wilson
Dr. Leo G. Egan	Philip C. Marriott	Paul A. Wolfgang
Joseph C. Evans	Roger Martin	Paul R. Work
John W. Fendrich	Tomoo Matsubara	Alfred W. Yonda
Roger G. Fordham	Scott D. Matthews	Janusz Zalewski

When the IEEE Standards Board approved this standard on Dec. 3, 1992, it had the following membership:

Marco W. Migliaro, *Chair*

Donald C. Loughry, *Vice Chair*

Andrew G. Salem, *Secretary*

Dennis Bodson	Donald N. Heirman	T. Don Michael*
Paul L. Borrill	Ben C. Johnson	John L. Rankine
Clyde Camp	Walter J. Karplus	Wallace S. Read
Donald C. Fleckenstein	Ivor N. Knight	Ronald H. Reimer
Jay Forster*	Joseph Koepfinger*	Gary S. Robinson
David F. Franklin	Irving Kolodny	Martin V. Schneider
Ramiro Garcia	D. N. "Jim" Logothetis	Terrance R. Whittemore
Thomas L. Hannan	Lawrence V. McCall	Donald W. Zipse
*Member Emeritus		

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall
Richard B. Engelman
David E. Soffrin
Stanley Warshaw

Rachel Auslander
IEEE Standards Project Editor

Contents

CLAUSE	PAGE
1. Overview.....	1
1.1 Scope.....	1
1.2 Audience	1
2. Definitions.....	2
3. Purpose of software quality metrics.....	4
4. Software quality metrics framework.....	5
5. The software quality metrics methodology.....	7
5.1 Establish software quality requirements	7
5.2 Identify software quality metrics	8
5.3 Implement the software quality metrics.....	10
5.4 Analyze the software metrics results	12
5.5 Validate the software quality metrics	13
ANNEXES	
Annex A Examples of factors, subfactors and metrics, and the relationships among them.....	19
Annex B Sample metrics descriptions	23
B.1 Detailed metrics descriptions.....	23
B.2 Sample metrics results	28
Annex C Examples of use of the methodology.....	33
C.1 Mission critical example	33
C.2 Example from a commercial environment.....	64
Annex D Annotated bibliography, bibliography, and standards and related documents	75
D.1 Annotated bibliography	75
D.2 Bibliography	87
D.3 Standards and related documents.....	88

IEEE Standard for a Software Quality Metrics Methodology

1. Overview

This standard is divided into five clauses. Clause 1 provides the scope of this standard. Clause 2 provides a set of definitions that are defined more fully in the standard, but are provided here as a quick reference. Clause 3 states the purpose of software quality metrics. Clause 4 provides a framework for software quality metrics. Clause 5 provides a methodology for software quality metrics, which is mandatory when applying this standard. Also in this standard are four annexes that are included for illustrative and reference purposes only.

1.1 Scope

This standard provides a methodology for establishing quality requirements and identifying, implementing, analyzing, and validating the process and product of software quality metrics. This methodology applies to all software at all phases of any software life cycle structure.

This standard does not prescribe specific metrics. However, the annexes include examples of metrics together with two complete examples of the use of this standard.

1.2 Audience

This standard is intended for those associated with the acquisition, development, use, support, maintenance, and audit of software. The standard is particularly aimed at those measuring or assessing the quality of software.

This standard can be used by

- An acquisition/project manager to identify, define, and prioritize the quality requirements for a system
- A system developer to identify specific traits that should be built into the software in order to meet the quality requirements
- A quality assurance/control/audit organization and a system developer to evaluate whether the quality requirements are being met
- A system maintainer to assist in change management during product evolution
- A user to assist in specifying the quality requirements for a system

2. Definitions

2.1 critical range: Metric values used to classify software into the categories of acceptable, marginal, or unacceptable.

2.2 critical value: Metric value of a validated metric that is used to identify software that has unacceptable quality.

2.3 direct metric: A metric applied during development or during operations that represents a software quality factor (for example, mean time to software failure for the factor reliability).

2.4 factor: *See: quality factor.* For the sake of brevity, the term “factor” is used in place of the term “quality factor” in this standard.

2.5 factor sample: A set of factor values that is drawn from the metrics database and used in metrics validation.

2.6 factor value: A value of the direct metric that represents a factor. (*See: metric value.*)

2.7 measure: A way to ascertain or appraise value by comparing it to a standard. To apply a metric.

2.8 measurement: The act or process of measuring. A figure, extent, or amount obtained by measuring.

2.9 metric: *See: software quality metric.* For the sake of brevity, the term “metric” is used in place of the term “software quality metric” in this standard.

2.10 metrics framework: A tool used for organizing, selecting, communicating, and evaluating the required quality attributes for a software system. A hierarchical breakdown of factors, subfactors, and metrics for a software system.

2.11 metrics sample: A set of metric values that is drawn from the metrics database and used in metrics validation.

2.12 metric validation: The act or process of ensuring that a metric correctly predicts or assesses a quality factor.

2.13 metric value: A metric output or an element that is from the range of a metric.

2.14 predictive assessment: The process of using a predictive metric(s) to predict the value of another metric.

2.15 predictive metric: A metric applied during development and used to predict the values of a software quality factor.

2.16 process step: Any task performed in the development, implementation, or maintenance of software (for example, identifying the software components of a system as part of the design).

2.17 process metric: A metric used to measure characteristics of the methods, techniques, and tools employed in developing, implementing, and maintaining the software system.

2.18 product metric: A metric used to measure the characteristics of the documentation and code.

2.19 quality attribute: A characteristic of software, or a generic term applying to factors, subfactors, or metric values.

2.20 quality factor: A management-oriented attribute of software that contributes to its quality.

2.21 quality requirement: A requirement that a software attribute be present in software to satisfy a contract, standard, specification, or other formally imposed document.

2.22 quality subfactor: A decomposition of a quality factor or quality subfactor to its technical components.

2.23 sample software: Software selected from a current or completed project from which data can be obtained for use in preliminary testing of data collection and metric computation procedures.

2.24 software component: A general term used to refer to a software system or an element, such as module, unit, data, or document.

2.25 software quality metric: A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.

NOTE—This definition differs from the definition of quality metric found in IEEE Std 610.12-1990.

2.26 subfactor: *See: quality subfactor.* For the sake of brevity, the term “subfactor” is used in place of the term “quality subfactor” in this standard.

2.27 validated metric: A metric whose values have been statistically associated with corresponding quality factor values.

3. Purpose of software quality metrics

Software quality is the degree to which software possesses a desired combination of attributes. This desired combination of attributes shall be clearly defined; otherwise, assessment of quality is left to intuition. For the purposes of this standard, defining software quality for a system is equivalent to defining a list of software quality attributes required for that system. In order to measure the software quality attributes an appropriate set of software metrics shall be identified.

The purpose of software metrics is to make assessments throughout the software life cycle as to whether the software quality requirements are being met. The use of software metrics reduces subjectivity in the assessment of software quality by providing a quantitative basis for making decisions about software quality. However, the use of software metrics does not eliminate the need for human judgment in software evaluations. The use of software metrics within an organization or project is expected to have a beneficial effect by making software quality more visible.

More specifically, the use of this standard's methodology for measuring quality allows an organization to

- Achieve quality goals
- Establish quality requirements for a system at its outset
- Establish acceptance criteria and standards
- Evaluate the level of quality achieved against the established requirements
- Detect anomalies or point to potential problems in the system
- Predict the level of quality that will be achieved in the future
- Monitor changes in quality when software is modified
- Assess the ease of change to the system during product evolution
- Normalize, scale, calibrate, or validate a metric

To accomplish these aims, both process and product metrics should be represented in the system metrics plan.

4. Software quality metrics framework

The software quality metrics framework shown in figure 1 is designed to be flexible. It permits additions, deletions, and modifications of factors, subfactors, and metrics. Each level may be expanded to several sub-levels. The framework can thus be applied to all systems and can be adapted as appropriate without changing the basic concept.

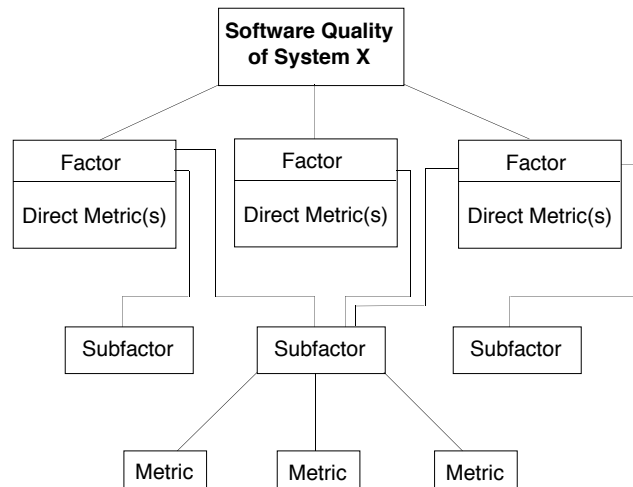


Figure 1—Software quality metrics framework

The first level of the software quality metrics framework hierarchy begins with the establishment of quality requirements by the assignment of various quality attributes. All attributes defining the quality requirements shall be agreed upon by the project team, and then the definitions should be established. Quality factors that represent management and user-oriented views are then assigned to the attributes. If necessary, subfactors are then assigned to each factor. Associated with each factor is a direct metric that serves as a quantitative representation of a quality factor. For example, a direct metric for the factor reliability could be mean time to failure. Each factor shall have one or more associated direct metrics and target values, such as one hour of execution time, that is set by project management. Otherwise, there is no way to determine whether the factor has been achieved. Some examples of factors are listed in annex A.

At the second level of the hierarchy are the quality subfactors that represent software-oriented attributes that indicate quality. These are obtained by decomposing each factor into measurable software attributes. Subfactors are independent attributes of software, and therefore may correspond to more than one factor (see annex A for further explanation). The subfactors are concrete attributes of software that are more meaningful than factors to technical personnel, such as analysts, designers, programmers, testers, and maintainers. The decomposition of factors into subfactors facilitates objective communication between the manager and the technical personnel regarding the quality objectives. Some examples of subfactors are listed in annex A.

At the third level of the hierarchy the subfactors are decomposed into metrics used to measure system products and processes during the development life cycle. Direct metric values, or factor values, are typically unavailable or expensive to collect early in the software life cycle. Therefore, metrics at the third level, that are validated against direct metrics, are used to estimate factor values early in the software life cycle.

From top to bottom the framework facilitates

- Establishment of quality requirements in terms of factors by managers early in a system's life cycle
- Communication of the established factors in terms of quality subfactors to the technical personnel
- Identification of metrics that are related to the established factors and subfactors

From bottom to top the framework enables the managerial and technical personnel to obtain feedback by

- Evaluating the software products and processes at the elementary metrics level
- Analyzing the metrics values to estimate and assess the quality factors

5. The software quality metrics methodology

The software quality metrics methodology is a systematic approach to establishing quality requirements and identifying, implementing, analyzing, and validating the process and product of software quality metrics for a software system. It spans the entire software life cycle and comprises five steps.

These steps shall be applied iteratively because insights gained from applying a step may show the need for further evaluation of the results of prior steps.

- a) *Establish software quality requirements.* A list of quality factors is selected, prioritized, and quantified at the outset of system development or system change. These requirements shall be used to guide and control the development of the system and, on delivery of the system, to assess whether the system met the quality requirements specified in the contract.
- b) *Identify software quality metrics.* The software quality metrics framework is applied in the selection of relevant metrics.
- c) *Implement the software quality metrics.* Tools are either procured or developed, data is collected, and metrics are applied at each phase of the software life cycle.
- d) *Analyze the software quality metrics results.* The metrics results are analyzed and reported to help control the development and assess the final product.
- e) *Validate the software quality metrics.* Predictive metrics results are compared to the direct metrics results to determine whether the predictive metrics accurately measure their associated factors.

The documentation or output produced as a result of these steps is shown in table 1.

Table 1 — Outputs of metrics methodology steps

Metric Methodology Step	Output
Establish software quality requirements	— Quality requirements
Identify software quality metrics	— Approved quality metrics framework — Metrics set — Cost-benefit analysis
Implement the software quality metrics	— Description of data items — Metrics/data item — Traceability matrix — Training plan and schedule
Analyze the software quality metrics results	— Organization and development process changes
Validate the software quality metrics	— Validation results

5.1 Establish software quality requirements

Quality requirements shall be represented in either of the following forms:

- *Direct metric value.* A numerical target for a factor to be met in the final product. For example, mean time to failure (MTTF) is a direct metric of final system reliability.
- *Predictive metric value.* A numerical target related to a factor to be met during system development. This is an intermediate requirement that is an early indicator of final system performance. For example, design or code errors may be early predictors of final system reliability.

5.1.1 Identify a list of possible quality requirements

Identify quality requirements that may be applicable to the software system. Use organizational experience, required standards, regulations, or laws to create this list. Annex A contains sample lists of factors and subfactors. In addition, list other system requirements that may affect the feasibility of the quality requirements. Consider acquisition concerns, such as cost or schedule constraints, warranties, and organizational self-interest. Do not rule out mutually exclusive requirements at this point. Focus on factor/direct metric combinations instead of predictive metrics.

All parties involved in the creation and use of the system shall participate in the quality requirements identification process.

5.1.2 Determine the actual list of quality requirements

Rate each of the listed quality requirements by importance. Importance is a function of the system characteristics and the viewpoints of the people involved. To determine the actual list of the possible quality requirements, follow the two steps below:

- *Survey all involved parties.* Discuss the relative priorities of the requirements with all involved parties. Have each group weigh the quality requirements against the other system requirements and constraints. Ensure that all viewpoints are considered.
- *Create the actual list of quality requirements.* Resolve the results of the survey into a single list of quality requirements. This shall involve a technical feasibility analysis of the quality requirements. The proposed factors for this list may have cooperative or conflicting relationships. Conflicts between requirements shall be resolved at this point. In addition, if the choice of quality requirements is in conflict with cost, schedule, or system functionality, one or the other shall be altered. Care shall be exercised in choosing the desired list to ensure that the requirements are technically feasible, reasonable, complementary, achievable, and verifiable. All involved parties shall agree to this final list.

5.1.3 Quantify each factor

For each factor, assign one or more direct metrics to represent the factor, and assign direct metric values to serve as quantitative requirements for that factor. For example, if “high efficiency” was one of the quality requirements from the previous item, the direct metric “actual resource utilization/allocated resource utilization” with a value of 90% could represent that factor. This direct metric value is used to verify the achievement of the quality requirement. Without it, there is no way to tell whether or not the delivered system meets its quality requirements.

The quantified list of quality requirements and their definitions again shall be approved by all involved parties.

5.2 Identify software quality metrics

When identifying software quality metrics, apply the software quality metrics framework, and perform a cost-benefit analysis. Then gain commitment to the metrics from all involved parties.

5.2.1 Apply the software quality metrics framework

Create a chart of the quality requirements based on the hierarchical tree structure found in figure 1. At this point, only the factor level must be complete. Next decompose each factor into subfactors as indicated in clause 4. The decomposition into subfactors must continue for as many levels as needed until the subfactor level is complete.

Using the software quality metrics framework, decompose the subfactors into measurable metrics. For each validated metric on the metric level, assign a target value and a critical value and range that should be achieved during development. The target values constitute additional quality requirements for the system.

The framework and the target values for the metrics shall be reviewed and approved by all involved parties.

To help ensure that metrics are used appropriately, only validated metrics (that is, either direct metrics or metrics validated with respect to direct metrics) shall be used to assess current and future product and process quality (see 5.5 for a description of the validation methodology and annex C for examples of applying the validation methodology). Nonvalidated metrics may be included for future analysis, but shall not be included as part of the system requirements. Furthermore, the metrics that are used shall be those that are associated with the quality requirements of the software project. However, given that the above conditions are satisfied, the selection of specific metrics as candidates for validation and the selection of specific validated metrics for application is at the discretion of the user of this standard. Examples of metrics and experiences with their use are given in annex B.

Document each metric using the format shown in table 2.

Table 2—Metrics set

Item	Description
Name	Name given to the metric.
Costs	Costs of using the metric (see 5.2.2.1).
Benefits	Benefits of using the metric (see 5.2.2.2).
Impact	Indication of whether a metric may be used to alter or halt the project (ask, <i>Can the metric be used to indicate deficient software quality?</i>).
Target value	Numerical value of the metric that is to be achieved in order to meet quality requirements. Include the critical value and the range of the metric.
Factors	Factors that are related to this metric.
Tools	Software or hardware tools that are used to gather and store data, compute the metric and analyze the results.
Application	Description of how the metric is used and what its area of application is.
Data items	Input values that are necessary for computing the metric values.
Computation	Explanation of the steps involved in the metrics computation.
Interpretation	Interpretation of the results of the metrics computation (see 5.4.1).
Considerations	Considerations of the appropriateness of the metric (for example, <i>Can data be collected for this metric? Is the metric appropriate for this application?</i>).
Training required	Training required to implement or use the metric.
Example	An example of applying the metric (see annex C).
Validation history	Names of projects that have used the metric, and the validity criteria the metric has satisfied.
References	References, such as list of projects, project details, etc., giving further details on understanding or implementing the metric.

5.2.2 Perform a cost-benefit analysis

Perform a cost-benefit analysis by identifying the costs of implementing the metrics, identifying the benefits of applying the metrics, and then applying the metrics set.

5.2.2.1 Identify the costs of implementing the metrics

Identify and document (see table 2) all the costs associated with the metrics in the metrics set. For each metric, estimate and document the following impacts and costs.

- *Metrics utilization costs.* The costs of collecting data; automating the metric value calculation (when possible); and analyzing, interpreting, and reporting the results.
- *Software development process change costs.* The set of metrics may imply a change in the development process.
- *Organizational structure change costs.* The set of metrics may imply a change in the organizational structure used to produce the software system.
- *Special equipment.* The hardware or software tools may have to be located, purchased, adapted, or developed to implement the metrics.
- *Training.* The quality assurance/control organization or the entire development team may need training in the use of the metrics and data collection procedures. If the introduction of metrics has caused changes in the development process, the development team may need to be educated about the changes.

5.2.2.2 Identify the benefits of applying the metrics

Identify and document the benefits that are associated with each metric in the metrics set (see table 2).

Some benefits to consider include the following:

- Identify quality goals and increase awareness of the goals in the software organization
- Provide timely feedback useful in developing higher quality software
- Increase customer satisfaction by quantifying the quality of the software before it is delivered to the customer
- Provide a quantitative basis for making decisions about software quality
- Reduce software life cycle costs by improving process efficiency based on metric data

5.2.2.3 Adjust the metrics set

Weigh the benefits, tangible and intangible, against the costs of each metric. If the costs exceed the benefits of a given metric, alter or delete it from the metrics set. On the other hand, for metrics that remain, make plans for any necessary changes to the software development process, organizational structure, tools, and training. In most cases it will not be feasible to quantify benefits. In these cases judgment shall be exercised in weighing qualitative benefits against quantitative costs.

5.2.3 Gain commitment to the metrics set

All involved parties shall review the revised metrics set to which the cost-benefit analysis has been added. The metrics set shall be formally adopted and supported by this group.

5.3 Implement the software quality metrics

To implement the software quality metrics, define the data collection procedures, use selected software to prototype the measurement process, then collect the data and compute the metrics values.

5.3.1 Define the data collection procedures

For each metric in the metric set, determine the data that will be collected and determine the assumptions that will be made about the data (for example, random sample, subjective or objective measure). The flow of data shall be shown from point of collection to evaluation of metrics. Describe or reference when and how tools are to be used and data storage procedures. Also, identify candidate tools. Select tools for use with the prototyping process. A traceability matrix shall be established between metrics and data items.

Identify the organizational entities that will directly participate in data collection including those responsible for monitoring data collection. Describe the training and experience required for data collection and the training process for personnel involved.

Describe each data item thoroughly, using the format shown in table 3.

Table 3—Description of a data item

Item	Description
Name	Name given to the data item.
Metrics	Metrics that are associated with the data item.
Definition	Straightforward description of the data item.
Source	Location of where the data originates.
Collector	Entity responsible for collecting the data.
Timing	Time(s) in life cycle at which the data is to be collected. (Some data items are collected more than once.)
Procedures	Methodology (for example, automated or manual) used to collect the data.
Storage	Location of where the data is stored.
Representation	Manner in which the data is represented, for example, its precision and format (e.g., Boolean, dimensionless, etc.).
Sample	Method used to select the data to be collected and the percentage of the available data that is to be collected.
Verification	Manner in which the collected data is to be checked for errors.
Alternatives	Methods that may be used to collect the data other than the preferred method.
Integrity	Who is authorized to alter this data item and under what conditions.

5.3.2 Prototype the measurement process

Test the data collection and metric computation procedures on selected software that will act as a prototype. If possible, the samples selected should be similar to the project(s) on which the metrics will later be used. An analysis shall be made to determine if the data is collected uniformly and if instructions have always been interpreted in the same manner. In particular, data requiring subjective judgments shall be checked to determine if the descriptions and instructions are clear enough to ensure uniform results.

In addition, the cost of the measurement process for the prototype shall be examined to verify or improve the cost analysis.

Use the results collected from the prototype to improve the metric descriptions (see table 2) and descriptions of data items (see table 3).

5.3.3 Collect the data and compute the metrics values

Using the formats in table 2 and table 3, collect and store data at the appropriate time in the life cycle. The data shall be checked for accuracy and proper unit of measure.

Data collection shall be monitored. If a sample of data is used, requirements such as randomness, minimum size of sample, and homogeneous sampling shall be verified. If more than one person is collecting the data, it shall be checked for uniformity.

Compute the metrics values from the collected data.

5.4 Analyze the software metrics results

Analyzing the results of the software metrics includes interpreting the results, identifying software quality, making software quality predictions, and ensuring compliance with requirements.

5.4.1 Interpret the results

The results shall be interpreted and recorded against the broad context of the project as well as for a particular product or process of the project. The differences between the collected metric data and the target values for the metrics shall be analyzed against the quality requirements. Substantive differences shall be investigated.

5.4.2 Identify software quality

Quality metric values for software components shall be determined and reviewed. Quality metric values that are outside the anticipated tolerance intervals (low or unexpected high quality) shall be identified for further study. Unacceptable quality may be manifested as excessive complexity, inadequate documentation, lack of traceability, or other undesirable attributes. The existence of such conditions is an indication that the software may not satisfy quality requirements when it becomes operational. Since many of the direct metrics that are usually of interest cannot be measured during software development (for example, reliability metrics), validated metrics shall be used when direct metrics are not available. Direct or validated metrics shall be measured for software components and process steps. The measurements shall be compared with critical values of the metrics. Software components whose measurements deviate from the critical values shall be analyzed in detail. The fact that a measurement deviates from a critical value does not necessarily mean that the software component will exhibit unacceptable quality during operation. Deviation from a critical value may occur because metrics are not infallible; they are only indicators of quality. What metrics indicate about quality during development may not be the quality achieved in operation. Depending on the results of the analysis, software components shall be redesigned (acceptable quality is achieved by redesign), scrapped (quality is so poor that redesign is not feasible), or not changed (deviations for critical metric values are judged to be insignificant).

Unexpected high quality metric values shall cause a review of the software development process as the expected tolerance levels may need to be modified or the process for identifying quality metric values may need to be improved.

5.4.3 Make software quality predictions

During development validated metrics shall be used to make predictions of direct metric values. Predicted values of direct metrics shall be compared with target values to determine whether to flag software components for detailed analysis. Predictions shall be made for software components and process steps. Software components and process steps whose predicted direct metric values deviate from the target values shall be analyzed in detail.

Potentially, prediction is very valuable because it estimates the metric of ultimate interest—the direct metric. However, prediction is difficult because it involves using validated metrics from an earlier phase of the life cycle (for example, development) to make a prediction about a different but related metric (direct metric) in a much later phase (for example, operations).

5.4.4 Ensure compliance with requirements

Direct metrics shall be used to ensure compliance of software products with quality requirements during system and acceptance testing (see 5.1). Direct metrics shall be measured for software components and process steps. These values shall be compared with target values of the direct metrics that represent quality requirements. Software components and process steps whose measurements deviate from the target values are non-compliant.

5.5 Validate the software quality metrics

For information on the statistical techniques used, see [B114], [B115], [B117],¹ or similar references.

5.5.1 Purpose of metrics validation

The purpose of metrics validation is to identify both product and process metrics that can predict specified quality factor values, which are quantitative representations of quality requirements. If metrics are to be useful, they shall indicate accurately whether quality requirements have been achieved or are likely to be achieved in the future. When it is possible to measure factor values at the desired point in the life cycle, these direct metrics are used to evaluate software quality. At some points in the life cycle, certain quality factor values (for example, reliability) are not available. They are obtained after delivery or late in the project. In these cases, other metrics are used early on in a project to predict quality factor values.

The history of the application of metrics indicates that predictive metrics were seldom validated (that is, it was not demonstrated through statistical analysis that the metrics measured software characteristics that they purported to measure.) However, it is important that predictive metrics be validated before they are used to evaluate software quality. Otherwise, metrics might be misapplied (that is, metrics might be used that have little or no relationship to the desired quality characteristics).

Although quality subfactors are useful when identifying and establishing factors and metrics, they need not be used in metrics validation, because the focus in validation is on determining whether a statistically significant relationship exists between predictive metric values and factor values.

Quality factors may be affected by multiple variables. A single metric, therefore, may not sufficiently represent any one factor if it ignores these other variables.

¹The numbers in brackets correspond to those of the bibliographical references in annex D.

5.5.2 Validity criteria

To be considered valid, a predictive metric shall demonstrate a high degree of association with the quality factors it represents. This is equivalent to accurately portraying the quality condition(s) of a product or process. A metric may be valid with respect to certain validity criteria and invalid with respect to other criteria.

The person in the organization who understands the consequence of the values selected shall designate threshold values for the following:

- V – square of the linear correlation coefficient
- B – rank correlation coefficient
- A – prediction error
- P – success rate

A short numerical example follows the definition of each validity criterion. Detailed examples of the application of metrics validation are contained in annex C.

- a) *Correlation.* The variation in the quality factor values explained by the variation in the metric values, which is given by the square of the linear correlation coefficient (R) between the metric and the corresponding factor, shall exceed V , where $R^2 > V$.

This criterion assesses whether there is a sufficiently strong linear association between a factor and a metric to warrant using the metric as a substitute for the factor, when it is infeasible to use the latter. For example, the correlation coefficient between a complexity metric and the factor reliability may be 0.8. The square of this is 0.64. Only 64% of the variation in the factor is explained by the variation in the metric. If V has been established as 0.7, the conclusion would be drawn that the metric is invalid (that is, there is insufficient association, or correlation between the metric and reliability). If this relationship is demonstrated over a representative sample of software components, the conclusion could be drawn that the metric is invalid.

- b) *Tracking.* If a metric M is directly related to a quality factor F , for a given product or process, then a change in a quality factor value from F_{T1} to F_{T2} , at times $T1$ and $T2$, shall be accompanied by a change in metric value from M_{T1} to M_{T2} , which is the same direction (for example, if F increases, M increases). If M is inversely related to F , then a change in F shall be accompanied by a change in M in the opposite direction (for example, if F increases, M decreases). To perform this test, compute the coefficient of rank correlation (r) from n paired values of the factor and the metric. Each of the factor/metric pairs is measured at the same point in time, and the n pairs of values are measured at n points in time. The absolute value of (r) shall exceed B .

This criterion assesses whether a metric is capable of tracking changes in product or process quality over the life cycle.

For example, if a complexity metric is claimed to be a measure of reliability, then it is reasonable to expect a change in the reliability of a software component to be accompanied by an appropriate change in metric value (for instance, if the product increases in reliability, the metric value should also change in a direction that indicates the product has improved). That is, if Mean Time to Failure (MTTF) is used to measure reliability and is equal to 1000 hours during testing ($T1$) and 1500 hours during operation ($T2$), a complexity metric whose value is 8 in $T1$ and 6 in $T2$, where 6 less complex than 8, is said to track reliability for this software component. If this relationship is demonstrated over a representative sample of software components, the conclusion could be drawn that the metric can track reliability (that is, indicate changes in product reliability) over the software life cycle.

- c) *Consistency.* If factor values $F1, F2, \dots, Fn$, corresponding to products or processes 1, 2, n , have the relationship $F1 > F2 > \dots, Fn$, then the corresponding metric values shall have the relationship $M1 > M2 > \dots, Mn$. To perform this test, compute the coefficient of rank correlation (r) between paired values (from the same software components) of the factor and the metric; $|r|$ shall exceed B .

This criterion assesses whether there is consistency between the ranks of the factor values of a set of software components and the ranks of the metric values for the same set of software components.

Thus, this criterion is used to determine whether a metric can accurately rank, by quality, a set of products or processes.

For example, if the reliability of software components *X*, *Y*, and *Z*, as measured by MTTF, is 1000, 1500, and 800 hours respectively, and the corresponding complexity metric values are 5, 3, and 7, where low metric values are better than high values, the ranks for reliability and metric values, with 1 representing the highest rank, are as follows:

<u>Software Component</u>	<u>Reliability Rank</u>	<u>Complexity Metric Rank</u>
<i>Y</i>	1	1
<i>X</i>	2	2
<i>Z</i>	3	3

If this relationship is demonstrated over a representative sample of software components, the conclusion could be drawn that the metric is consistent and can be used to rank the quality of software components. For example, the ranks could be used to establish priority of testing and allocation of budget and effort to testing (that is, the worst software component would receive the most attention, largest budget, and most staff).

- d) *Predictability*. If a metric is used at time T_1 to predict a quality factor for a given product or process, it shall predict a related quality factor Fp_{T_2} with an accuracy of

$$\left| \frac{Fa_{T_2} - Fp_{T_2}}{Fa_{T_2}} \right| < A$$

where Fa_{T_2} is the actual value of F at time T_2 .

This criterion assesses whether a metric is capable of predicting a factor value with the required accuracy.

For example, if a complexity metric is used during development to predict the reliability of a software component during operation (T_2) to be 1200 hours MTTF (Fp_{T_2}) and the actual MTTF that is measured during operation is 1000 hours (Fa_{T_2}), then the error in prediction is 200 hours, or 20%. If the acceptable prediction error (A) is 25%, prediction accuracy is acceptable. If the ability to predict is demonstrated over a representative sample of software components, the conclusion could be drawn that the metric can be used as a predictor of reliability. For example, prediction could be used during development to identify those software components that need to be improved.

- e) *Discriminative Power*. A metric shall be able to discriminate between high quality software components (for example, high MTTF) and low quality software components (for example, low MTTF). For instance, the set of metric values associated with the former should be significantly higher (or lower) than those associated with the latter.

This criterion assesses whether a metric is capable of separating a set of high quality software components from a set of low quality components. This capability identifies critical values for metrics that can be used to identify software components that may have unacceptable quality. The Mann-Whitney Test and the Chi-square test for differences in probabilities (Contingency Tables) can be used for this validation test.

For example, if all software components with a complexity metric value of greater than 10 (critical value) have a MTTF of 1000 hours and all components with a complexity metric value equal to or less than 10 have a MTTF of 2000 hours, and this difference is sufficient to pass the statistical tests, then the metric separates low from high quality software components. If the ability to discriminate is demonstrated over a representative sample of software components, the conclusion could be drawn that the metric can discriminate between low and high reliability components for quality assurance and other quality functions.

- f) *Reliability*. A metric shall demonstrate the above correlation, tracking, consistency, predictability, and discriminative power properties for P percent of the applications of the metric.

This criterion is used to ensure that a metric has passed a validity test over a sufficient number or percentage of applications so that there will be confidence that the metric can perform its intended function consistently.

For example, if the required success rate (P) for validating a complexity metric against the predictability criterion has been established as 80%, and there are 100 software components, the metric shall predict the factor with the required accuracy for at least 80 of the components.

5.5.3 Validation procedure

Metrics validation shall include identifying the quality factors sample, identifying the metrics sample, performing a statistical analysis, and documenting the results.

5.5.3.1 Identify the quality factors sample

These factor values (for example, measurements of reliability), which represent the quality requirements of a project, were previously identified (see 5.1.3) and collected and stored (see 5.3.3). For validation purposes, draw a sample from the metrics database.

5.5.3.2 Identify the metrics sample

These metrics (for example, design structure) are used to predict or to represent quality factor values, when the factor values cannot be measured. The metrics were previously identified (see 5.2.1) and their data collected and stored, and values computed from the collected data (see 5.3.3). For validation purposes, draw a sample from the same domain (for example, same software components) of the metrics database as used in 5.5.3.1.

5.5.3.3 Perform a statistical analysis

The tests described in 5.5.2 shall be performed.

Before a metric is used to evaluate the quality of a product or process, it shall be validated against the criteria described in 5.5.2. If a metric does not pass all of the validity tests, it shall only be used according to the criteria prescribed by those tests (for example, if it only passes the tracking validity test, it shall be used only for tracking the quality of a product or process).

5.5.3.4 Document the results

Documenting results shall include the direct metric, predictive metric, validation criteria, and numerical results, as a minimum.

5.5.4 Additional requirements

Additional requirements, those being the need for revalidation, confidence in analysis results, and the stability of environment, are described in the following subclauses.

5.5.4.1 The need for revalidation

It is important to revalidate a predictive metric before it is used for another environment or application. As the software engineering process changes, the validity of metrics changes. Cumulative metric validation values may be misleading because a metric that has been valid for several uses may become invalid. It is wise to compare the one-time validation of a metric with its validation history to avoid being misled.

The following statements of caution should be noted:

- A validated metric may not necessarily be valid in other environments or future applications.
- A metric that has been invalidated may be valid in other environments or future applications.

5.5.4.2 Confidence in analysis results

Metrics validation is a continuous process. Confidence in metrics is increased over time as metrics are validated on a variety of projects and as the metrics database and sample size increases. Confidence is not a static, one-time property. If a metric is valid, confidence will increase with increased use (that is, the correlation coefficient will be significant at decreasing values of the significance level). Greatest confidence occurs when metrics have been tentatively validated based on data collected from previous projects. Even when this is the case, the validation analysis will continue into future projects as the metrics database and sample size grow.

5.5.4.3 Stability of environment

Practicable metrics validation shall be undertaken in a stable development environment (that is, where the design language, implementation language, or program development tools do not change over the life of the project in which validation is performed). In addition, there shall be at least one project in which metrics data have been collected and validated prior to application of the predictive metrics. This project shall be similar to the one in which the metrics are applied with respect to software engineering skills, application, size, and software engineering environment.

Validation and application of metrics shall be performed during the same life cycle phases on different projects. For example: if metric *X* is collected during the development phase of project *A* and the saved values are later validated with respect to direct metric *Y*, whose values are collected during the operations phase of project *A*, the metric *X* shall be used during the development phase of project *B* to assess direct metric *Y* with respect to the operations phase of project *B*.

Annexes

(These informative annexes are not a part of IEEE Std 1061-1992, IEEE Standard for a Software Quality Metrics Methodology, but are included for information only.)

Annex A Examples of factors, subfactors and metrics, and the relationships among them

(informative)

Tables A1 and A2 illustrate descriptions of factors and subfactors, respectively. This selection was made to provide illustrative and usable definitions of factors and subfactors. Figure A1 illustrates a possible set of metrics for each one of the factors and its associated subfactors.

Note that these tables are not exhaustive. There are many possible sets of factors, subfactors, and metrics that can be used. Consult annex D for additional sources of this information.

Table A1—Sample factors

Factor	Description
Efficiency	An attribute that bears on the relationship of the level of performance to the amount of resources used under stated conditions.
Functionality	An attribute that bears on the existence of certain properties and functions that satisfy stated or implied needs of users.
Maintainability	An attribute that bears on the effort needed for specific modifications.
Portability	An attribute that bears on the ability of software to be transferred from one environment to another.
Reliability	An attribute that bears on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
Usability	An attribute that bears on the effort needed for use (including preparation for use and evaluation of results) and on the individual assessment of such use by users.

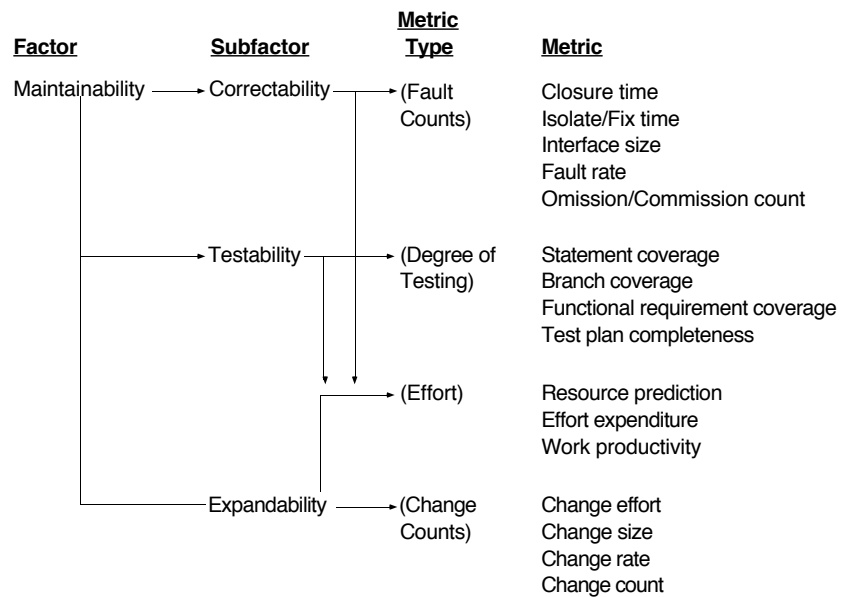
Table A2—Sample subfactors

Factor	Subfactor	Description
<i>Efficiency</i>	Time economy	Capability of software to perform specified functions under stated or implied conditions within appropriate time frames.
	Resource economy	Capability of software to perform specified functions under stated or implied conditions, using appropriate amounts of resources.
<i>Functionality</i>	Completeness	The degree to which software possesses necessary and sufficient functions to satisfy user needs.
	Correctness	The degree to which all software functions are specified.
	Security	The degree to which software can detect and prevent information leak, information loss, illegal use, and system resource destruction.
	Compatibility	The degree to which new software can be installed without changing environments and conditions that were prepared for the replaced software.
	Interoperability	The degree to which software can be connected easily with other systems and operated.
<i>Maintainability</i>	Correctability	The degree of effort required to correct errors in software and cope with user complaints.
	Expandability	The degree of effort required to improve or modify the efficiency or functions of software.
	Testability	The effort required to test software.
<i>Portability</i>	Hardware independence	The degree to which software does not depend on specific hardware environments.
	Software independence	The degree to which software does not depend on specific software environments.
	Installability	The effort required to adjust software to a new environment.
	Reusability	The degree to which software can be reused in applications other than the original application.
<i>Reliability</i>	Nondeficiency	The degree to which software does not contain undetected errors.
	Error tolerance	The degree to which software will continue to work without a system failure that would cause damage to the users. Also, the degree to which software includes degraded operation and recovery functions.
	Availability	The degree to which software remains operable in the presence of system failures.

Table A2—Sample subfactors (Continued)

Factor	Subfactor	Description
<i>Usability</i>	Understandability	The amount of user effort required to understand software.
	Ease of learning	The degree to which user effort required to understand software is minimized.
	Operability	The degree to which the operation of software matches the purpose, environment, and physiological characteristics of users, including ergonomic factors such as color, shape, sound, etc.
	Communicativeness	The degree to which software is designed in accordance with the psychological characteristics of users.

Figure A1 maps a factor to subfactors to metrics. This is for illustrative purposes only. Note that, in general, mappings between factors, subfactors, and metrics may be one-to-one, one-to-many, or many-to-one.

**Figure A1—Sample diagram to show the relationships between factors, subfactors, and some metrics**

Annex B

Sample metrics descriptions

(informative)

This annex provides the following:

- a) Examples of how to compute software metrics
- b) Detailed descriptions of some major metrics
- c) Sample metric results

The examples are not intended to be all inclusive. Additional examples exist but, due to space limitations, all of them cannot be documented here.

B.1 Detailed metrics descriptions

The following summarizes some major metrics. The references listed below should be consulted for a complete understanding of the metrics.

Table B1 — Example metrics computations

Factor/Subfactor	Computation
Completeness	Ratio of number of completed documents or software components to total number of documents or software components.
Consistency	Ratio of number of documents or software components that are free of contradictions to total number of documents or software components.
Correctness	Ratio of number of user mission objectives that have been correctly implemented in software components to total number of user mission objectives.
Interoperability	Ratio of number of interface specifications or software components whose interfaces are compatible to total number of interface specifications or software components.
Maintainability	Ratio of total error correction labor time to total number of corrections.
Reliability	Ratio of number of computer runs that produce correct outputs to total number of computer runs.
Testability	Identification of independent paths corresponding to McCabe metric for path testing.
Traceability	Ratio of number of documents or software components for this phase that can be traced to the previous phase to total number of documents or software components for this phase.