IEEE

IEEE Standard Dictionary of Measures of the Software Aspects of Dependability

IEEE Computer Society

Sponsored by the
Software Engineering Standards Committee

982.1™

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

8 May 2006

IEEE Std 982.1™-2005
(Revision of
IEEE Std 982.1-1988)

IEEE Std 982.1™-2005
(Revision of
IEEE Std 982.1-1988)

IEEE Standard Dictionary of Measures of the Software Aspects of Dependability

Sponsor
Software Engineering Standards Committee
of the
IEEE Computer Society

Approved 8 November 2005

IEEE-SA Standards Board

Abstract: A standard dictionary of measures of the software aspects of dependability for assessing and predicting the reliability, maintainability, and availability of any software system; in particular, it applies to mission critical software systems.
Keywords: availability, dependability, maintainability, reliability

---

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Print: ISBN 0-7381-4846-6 SH95392
PDF: ISBN 0-7381-4847-4 SS95392

Introduction

This introduction is not part of IEEE Std 982.1-2005, IEEE Standard Dictionary of Measures of the Software Aspects of Dependability.

Rationale for revision: In accordance with Software Engineering Standards Committee (SESC) policy, the first standard (this document) is a small document with just a few core measures dealing with reliability, maintainability, and availability. This will be followed by a second standard that will address safety, confidentially, and integrity.

This standard has not been revised since 1988. It was reaffirmed in 1996, but there were significant negative comments. It has been revised because many of the original measures had the following undesirable characteristics:

— Unrealistic (i.e., there was naivety about the necessary data, personnel capabilities, and training to effectively use the measures)

— Did not measure what they purported to measure

— Complex equations that were hard to understand and implement

— Immature (i.e., they were not widely used)

— Little field data to back up claims for benefits

History

This standard is 15 years old, and additional information on the topic has been developed since that time. This project updated the standard to include such information and to expand the scope of the standard. Specifically, the following topics were included in the standard:

a) Determined specific criteria for inclusion of measures in the standard

b) Performed a measure-by-measure review of the measures in the standard, with the following goals:

    1) Determined whether the measure should be modified, retained, or deleted, based on the criteria. The modified and retained measures are identified in this standard by their numbers from the original IEEE 982.1 standard (i.e., IEEE 982 #X).

    2) Added any additional information on the measures developed since 1988.

    3) Clarified definition and implementation conventions.

    4) Identified and incorporated, where appropriate, new measures that have appeared since 1988.

    5) Formulated generic measure classes and categorized the measures into these classes. These classes are reliability, maintainability, and availability.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Participants

At the time this standard was completed, the P982 Working Group had the following membership:

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Contents

IEEE Standard Dictionary of Measures of the Software Aspects of Dependability

1. Overview

1.1 General

This standard has the following clauses: Clause 1 provides the scope of this standard. Clause 2 provides a set of definitions. Clause 3 through Clause 5 specify new, modified, and existing measures for reliability, respectively. Clause 6 and Clause 7 specify new measures for maintainability and availability, respectively. In addition, there are three informative annexes. Annex A documents the process that was used to determine whether a measure in the existing standard should be modified, retained, or deleted; this annex states the criteria that were used to determine whether a measure should be included in the standard, including new measures. Annex B describes the *Software Reliability Engineering Case Study*, which provides a case study example of how to apply selected measures in Clause 3. Annex C is a glossary of terms taken from *The Authoritative Dictionary of IEEE Standards Terms* [B8].[1] Annex D is a bibliography that provides additional information about measures in Clause 3 through Clause 7. Table 1 summarizes the categories of measures. Numerical applications of the measures can be found in the reference(s) that are keyed to each measure.

[1] The numbers in brackets correspond to those of the bibliography in Annex D.

Table 1—Summary of measures

Identification	Name
Clause 3. New reliability measures	
3.2	Time to next failure (*s*)
3.3	Risk factor regression model
3.4	Remaining failures
3.5	Total test time to achieve specified remaining failures
3.6	Network reliability
Clause 4. Modified reliability measures	
4.1	Defect density (982 #2)
4.2	Test coverage index (982 #5)
4.3	Requirements compliance (982 #23)
4.4	Failure rate (982 #31)
Clause 5. Retained reliability measures	
5.1	Fault density (982 #1)
5.2	Requirements traceability (982 #7)
5.3	Mean time to failure (MTTF) (982 #30)
Clause 6. New maintainability measures	
6.1	Mean time to repair
6.2	Network maintainability
Clause 7. New availability measures	
7.1	Availability
7.2	Network availability

1.2 Scope

This standard specifies and classifies measures of the software aspects of dependability. It is an expansion of the scope of the existing standard; the revision includes the following aspects of dependability: reliability, availability, and maintainability of software. The applicability of this standard is any software system; in particular, it applies to mission critical systems, where high reliability, availability, and maintainability are of utmost importance. These systems include, but are not limited to, systems for military combat, space missions, air traffic control, network operations, stock exchanges, automatic teller machines, and airline reservation systems.

1.3 Purpose

This standard provides measures that are applicable for continual self-assessment and improvement of the software aspects of dependability.

1.4 Compliance

An application of the measures specified herein complies with this standard if Clause 2 through Clause 7 are implemented, with the exception of subclauses identified as either “Experience” and “Tools”. These subclauses are intended to (1) provide information about application domains for a given measure and (2) identify the types of tools appropriate for computing that measure.

1.5 Interaction with other standards

The measures defined in this dictionary may provide information that can help support certain evaluations discussed in other IEEE Software Engineering Standards. The following list is meant to be suggestive:

IEEE/EIA Std 12207.0
Subclause 6.4.2 of IEEE/EIA Std 12207.0 lists verification tasks, and 6.5 lists validation tasks. Dependability measures may provide useful quantitative information that will help satisfy these tasks. For example, in 6.4.2.5, one criterion states (among other requirements) that "the code implements … correct data and control flow."[1]

IEEE Std 1074™
Subclause A.5.1 discusses evaluation activities, including reviews, traceability, testing, and reporting evaluation results. Dependability measures can provide useful quantitative information that can be used to assist these activities.[2, 3]

IEEE Std 1012™
This standard lists verification and validation tasks, inputs, and outputs in Table 1. The dependability measures provided in this standard can provide useful quantitative information that can assist many of the tasks listed in this table.

2. Definitions

For the purposes of this document, the following terms and definitions apply. The glossary in Annex C and *The Authoritative Dictionary of IEEE Standards Terms* [B8] should be referenced for terms not defined in this clause.

2.1 defect: A generic term that can refer to either a fault (cause) or a failure (effect). (adapted from Lyu [B12])

2.2 dependability: Trustworthiness of a computer system such that reliance can be justifiably placed on the service it delivers. Reliability, availability, and maintainability are aspects of dependability. (adapted from Lyu [B12])

2.3 maintainability: Speed and ease with which a program can be corrected or changed. (adapted from Musa [B14])

2.4 measure (noun): (**A**) The number or symbol assigned to an entity by a mapping from the empirical world to the formal, relational world in order to characterize an attribute. (adapted from Fenton and Pfleeger [B5]) (**B**) The act or process of measuring. (adapted from *Webster's New Collegiate Dictionary* [B23])

2.5 measure (verb): To make a measurement. (adapted from *Webster's New Collegiate Dictionary* [B23])

2.6 time: In decreasing order of resolution, CPU execution time, elapsed time (i.e., wall clock time), or calendar time. (adapted from Musa [B14])

1 EIA publications are available from Global Engineering Documents, 15 Inverness Way East, Englewood, CO 80112, USA (http://global.ihs.com/).
2 IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).
3 The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

3. New reliability measures

3.1 General

The software reliability model specified in this clause (Schneidewind [B20]) is one of the four models recommended in ANSI R-013-1992 [B1]. See this reference for the other three recommended models that could be used in place of the specified model.

3.2 Time to next failure (*s*) (Lyu [B12])

3.2.1 Definition

$T_F\ (t)$ is the predicted time for the next F_t failures to occur, when the current time is t.

$$T_F\ (t) = [(\log[\alpha/(\alpha - \beta(X_{s,t} + F_t))])/] - (t - s + 1)$$

3.2.2 Variables

$T_F\ (t)$ is the predicted time for the next F_t failures to occur, when the current time is t.

$X_{s,t}$ is the observed failure count in the range $[s,t]$.

T is the test or operational time interval when *time to next failure* prediction is made.

3.2.3 Parameters

α is the failure rate at the beginning of interval s.

β is the negative of derivative of failure rate divided by failure rate.

F_t is the given number of failures to occur after interval t.

s is the starting interval for using observed failure data in parameter estimation.

t_m is the mission duration.

3.2.4 Application

Given a mission duration requirement t_m and number of failures F_t, a prediction is made at time t to see whether $T_F(t) > t_m$.

3.2.5 Data requirements

Failure counts by time interval, in the range $[1,t]$, where time can be execution time, clock time, or calendar time, for estimating α, β, and s, and for obtaining $X_{s,t}$.

3.2.6 Units of measurement

$T_F\ (t)$, t, t_m is the seconds, minutes, hours, and so on, as appropriate.

F_t is the number of failures.

3.2.7 Experience

Space Shuttle Primary Avionics Software Subsystem, United States Navy Tomahawk cruise missile launch, readiness validation for Trident nuclear missile, and United States Marine Corps Tactical Systems Support Activity for distributed system software reliability assessment and prediction (Keller and Schneidewind [B11]).

3.2.8 Tools

Software reliability modeling and analysis tools. Examples of these tools are described in Appendix A of Lyu [B12].

3.3 Risk factor regression model (Schneidewind [B22])

3.3.1 Definitions

CF is the cumulative *memory space* reliability prediction equation:

$$CF = a \times CS^2 - b \times CS + c$$

CF is the cumulative *requirements issues* reliability prediction equation:

$$CF = d \times (\exp (e \times CI))$$

RF (risk factor) is the attributes of a requirements change that can induce reliability risk, such as *memory space* and *requirements issues*.

“Memory space” is the amount of memory space required to implement a requirements change (i.e., a requirements change uses memory to the extent that other functions do not have sufficient memory to operate effectively, and failures occur).

“Requirements issues” is the number of conflicting requirements (i.e., a requirements change conflicts with another requirements change, such as requirements to increase the search criteria of a website and simultaneously decrease its search time, with the added software complexity causing failures).

3.3.2 Variables

CF is the cumulative failure (over a set of requirements changes).

CI is the cumulative issue (over a set of requirements changes).

CS is the cumulative space (over a set of requirements changes).

3.3.3 Parameters

Coefficients of nonlinear regression equations: *a*, *b*, *c*, *d*, and *e*.

3.3.4 Application

Provide a warning to software managers of impending reliability problems early in the development cycle, during requirements analysis, by using risk factors to predict cumulative failures and the values of the risk factor thresholds where reliability would degrade significantly. Thus, software managers could anticipate problems rather than react to them. In addition, more efficient software management would be possible because with advance warning of reliability problems, management could better schedule and prioritize development process activities (e.g., inspections, tests).

3.3.5 Data requirements

CF is the cumulative failure count.

CI is the cumulative requirements issues count.

CS is the cumulative memory space count.

3.3.6 Units of measurement

CF, CI are the cumulative failure count and cumulative requirements issues count: dimensionless numbers.

CS is the cumulative memory space requirement count in suitable units (e.g., words).

3.3.7 Experience

Space Shuttle Primary Avionics Software Subsystem (Schneidewind [B22]).

3.3.8 Tools

Spreadsheet or statistical analysis software.

3.4 Remaining failures (Keller and Schneidewind [B11])

3.4.1 Definitions

$r(t_\mathrm{t})$ are the remaining failures predicted at total test time t_t:

$$r(t_\mathrm{t}) = (\alpha/\beta)(\exp - \beta[t_\mathrm{t} - (s-1)])$$

3.4.2 Variables

$r(t_t)$ are the remaining failures.

t_t is the total test time of a release or module.

3.4.3 Parameters

α is the failure rate at the beginning of interval s.

β is the negative of derivative of failure rate divided by failure rate.

s is the starting interval for using observed failure data in parameter estimation.

r_c is the critical value of remaining failures, specified according to the criticality of the application.

3.4.4 Application

Test whether the safety criterion, predicted remaining failures $r(t_t) < r_c$, is met.

3.4.5 Data requirements

Failure counts by time interval, in the range $[1, t_t]$, where time can be execution time, clock time, or calendar time, for estimating α, β, and s.

t_t is the total test time of a release or module.

3.4.6 Units of measurement

$r(t_t)$ is the remaining failures: failure count.

t_t is the total test time of a release or module: seconds, minutes, hours, and so on, as appropriate.

r_c is the critical value of remaining failures: number of failures.

3.4.7 Experience

Space Shuttle Primary Avionics Software Subsystem (Keller and Schneidewind [B11]).

3.4.8 Tools

Software reliability modeling and analysis tools, spreadsheet software.

3.5 Total test time to achieve specified remaining failures (Schneidewind [B20])

3.5.1 Definition

t_t is the predicted total time from the start of test required to achieve a specified number of remaining failures $r(t_t)$:

$$t_t = [\log[\alpha/(\beta[r(t_t)])]]/\beta + (s - 1)$$

3.5.2 Variables

t_t is the predicted total test time for a release or module.

3.5.3 Parameters

$r(t_t)$ is the specified number of remaining failures for predicting t_t.

α is the failure rate at the beginning of interval s.

β is the negative of derivative of failure rate divided by failure rate.

S is the starting interval for using observed failure data in parameter estimation.

3.5.4 Application

Predict total test time as a function of the reliability goal, as specified by number of remaining failures.

3.5.5 Data requirements

Failure counts by time interval, in the range [1, t_t], where time can be execution time, clock time, or calendar time, for estimating α, β, and s.

$r(t_t)$ is the specified number of remaining failures.

3.5.6 Units of measurement

$r(t_t)$ is the failure count.

t_t are the seconds, minutes, hours, and so on, as appropriate.

3.5.7 Experience

Space Shuttle Primary Avionics Software Subsystem, United States Navy Tomahawk cruise missile launch, readiness validation for Trident nuclear missile, and United States Marine Corps Tactical Systems Support Activity for distributed system software reliability assessment and prediction (Keller and Schneidewind [B11]).

3.5.8 Tools

Software reliability modeling and analysis tools, spreadsheet software.

3.6 Network reliability (Schneidewind [B19])

3.6.1 Definition

RN is the reliability of network (probability of network surviving).

$$RN = RC \times RS$$

3.6.2 Variables

RC is the reliability of nc clients (probability of all clients surviving).

$$RC = 1 - \frac{\sum_{i=1}^{nc} TC_i}{nc \times T}$$

RS is the reliability of ns servers (probability of all servers surviving).

$$RS = 1 - \frac{\sum_{i=1}^{ns} TS_i}{ns \times T}$$

TC_i is the downtime on client i (down due to software failure and repair) during time T.

TS_i is the downtime on server i (down due to software failure and repair) during time T.

3.6.3 Parameters

T is the scheduled operational time.

nc is the number of client nodes in network.

ns is the number of server nodes in network.

3.6.4 Application

Reliability assessment of networks.

3.6.5 Data requirements

TC_i is the downtime of client i during time T.

TS_i is the downtime of server i during time T.

T is the scheduled operational time.

nc is the number of client nodes in network.

ns is the number of server nodes in network.

3.6.6 Units of measurement

TC_i, TS_i, T are the minutes, hours, and days, as appropriate.

nc, ns are the dimensionless numbers.

3.6.7 Experience

United States Marine Corps Tactical Systems Support Activity for distributed system software reliability assessment and prediction (Schneidewind [B19]).

3.6.8 Tools

Spreadsheet software.

4. Modified reliability measures

4.1 Defect density (982 #2) (Fenton and Pfleeger [B5] and Nikora et al. [B17])

4.1.1 Definition

DD is the defect density.

$$DD = \frac{D}{KSLOC}$$

4.1.2 Variables

D is the number of defects (or discrepancy reports) of a specified severity, per release or module.

KSLOC is the number of source lines of executable code and non-executable data declarations in thousands per release or module.

4.1.3 Parameters

Specified severity level.

4.1.4 Application

Track software quality across a series of releases or modules.

4.1.5 Data requirements

D is the count of defects (or discrepancy reports) of a specified severity, per release or module.

KSLOC is the count of number of source lines of executable code and non-executable data declarations in thousands per release or module.

4.1.6 Units of measurement

Dimensionless numbers.

4.1.7 Experience

Space Shuttle Primary Avionics Software Subsystem (Nikora and Munson [B16]).

4.1.8 Tools

Spreadsheet software, compiler output of source lines of executable code and non-executable data declarations.

4.2 Test coverage index (982 #5) (Binder [B2])

4.2.1 Definition

TCI is the test coverage index

$$TCI = \frac{NR}{TR}$$

4.2.2 Variables

NR is the number of requirements that have passed all tests per release or module.

TR is the total number of requirements tested per release or module.

4.2.3 Parameters

None.

4.2.4 Application

Determine whether requirements have been covered in testing.

4.2.5 Data requirements

NR is the number of requirements that have passed all tests per release or module.

TR is the total number of requirements tested per release or module.

4.2.6 Units of measurement

NR, *TR* are the dimensionless numbers.

4.2.7 Experience

Commercial software development organization test procedures (Lyu [B12]).

4.2.8 Tools

Spreadsheet software, data management system for keeping track of requirements and test results.

4.3 Requirements compliance (982 #23) (Fischer and Walker [B6])

4.3.1 Definition

Percentage of errors due to inconsistent requirements = $N_1/(N_1 + N_2 + N_3) \times 100$

Percentage of errors due to incomplete requirements = $N_2/(N_1 + N_2 + N_3) \times 100$

Percentage of errors due to misinterpreted requirements = $N_3/(N_1 + N_2 + N_3) \times 100$

4.3.2 Variables

N_1 is the number of inconsistent requirements in a release or module.

N_2 is the number of incomplete requirements in a release or module.

N_3 is the number of misinterpreted requirements in a release or module.

4.3.3 Parameters

None.

4.3.4 Application

An analysis is made of the percentages for the respective requirements error types: inconsistencies, incompleteness, and misinterpretation. Identification of inconsistent, incomplete, and misinterpreted requirements. Requirements problems can have an adverse effect on reliability.

4.3.5 Data requirements

N_1 is the count of inconsistent software requirements in a release or module, which is the number of decomposition elements that do not accurately reflect the system requirement specification.

N_2 is the count of incomplete software requirements in a release or module, which is the number of decomposition elements that do not completely reflect the system requirement specification.

N_3 is the count of misinterpreted software requirements in a release or module, which is the number of decomposition elements that do not correctly reflect the system requirement specification.

4.3.6 Units of measurement

N_1, N_2, N_3 are the dimensionless numbers.

4.3.7 Experience (Fischer and Walker [B6])

Shipboard tactical command center, combat weapons system, and communication system; ground-based tactical command center; and air traffic control.

4.3.8 Tools

Spreadsheet software.

4.4 Failure rate (982 #31) (Lyu [B12])

4.4.1 Definition

f is the incremental failure count/incremental test or operational time.

$f = \Delta f_i/\Delta t_i$ is the incremental failure count/incremental test or operational time.

4.4.2 Variables

Δt_i is the test or operational time expended in period i by a release or module.

Δf_i is the number of failures that occur during Δt_i in a release or module.

4.4.3 Parameters

i is the period identifier.

n is the number of periods.

4.4.4 Application

Track software reliability over the periods 1, 2, 3, … , i, … , n to see whether it is increasing (i.e., decreasing failure rate) or decreasing (i.e., increasing failure rate).

4.4.5 Data requirements

Δt_i is the test or operational time expended in period i (e.g., 30 minutes in hour 1) by a release or module.

Δf_i is the number of failures that occur during Δt_i (e.g., two failures in 30 minutes in hour 1) in a release or module.

4.4.6 Units of measurement

Δt_i is the test or operational time expended in period i: seconds, minutes, hours, and so on, as appropriate, by a release or module.

Δf_i is the number of failures that occur during Δt_i in a release or module.

4.4.7 Experience

Military distributed processing system (Lyu [B12]).

4.4.8 Tools

Software reliability modeling and analysis tools, spreadsheet software.

5. Retained reliability measures

5.1 Fault density (982 #1) (Musa [B14] and Nikora and Munson [B16])

5.1.1 Definitions

FD is the fault density.

$$FD = \frac{F}{KSLOC}$$

5.1.2 Variables

FD is the fault density.

F is the number of unique faults found that resulted in failures of a specified severity level, per release or module.

KSLOC is the number of source lines of executable code and non-executable data declarations in thousands, per release or module.

5.1.3 Parameters

Specified severity level.

5.1.4 Application

This measure is used to perform the following functions:

a) Establish target fault density by severity class.

b) Compare calculated fault density with target value by severity class.

c) Use function b) to determine whether sufficient testing has been completed.

5.1.5 Data requirements

F is the count of number of unique faults found that resulted in failures of a specified severity level, per release or module.

KSLOC is the count of number of source lines of executable code and non-executable data declarations in thousands, per release or module.

5.1.6 Units of measurement

F, *KSLOC* are the dimensionless numbers.

5.1.7 Experience

Space Shuttle Primary Avionics Software Subsystem (Nikora and Munson [B16]).

5.1.8 Tools

Spreadshset software, compiler output of source lines of executable code and non-executable data declarations.

5.2 Requirements traceability (982 #7) (Fenton and Pfleeger [B5])

5.2.1 Definition

TM is the traceability measure.

$$TM = \left(\frac{R1}{R2}\right) \times 100\%$$

If $TM \leq 100\%$, then no excess requirements are implemented in the release or module;
Else there are excess requirements in the amount ($TM - 100$)%.

5.2.2 Variables

TM is the traceability measure.

R1 is the number of requirements implemented in the release or module.

R2 is the number of original requirements specified for the release or module.

5.2.3 Parameters

None.

5.2.4 Application

This measure aids in identifying requirements that are either missing from, or in addition to, the original requirements. Missing requirements negatively affect reliability. Excess requirements negatively affect the software development budget.

A set of mappings from the requirements in the software implementation to the original requirements is created. Count each requirement met by the implementation ($R1$), and count each of the original requirements ($R2$). Compute the traceability measure:

$$TM = \left(\frac{R1}{R2}\right) \times \ 100\%$$

5.2.5 Data requirements

R1 is the count of the requirements implemented in the release or module.

R2 is the count of the original requirements specified in the release or module.

5.2.6 Units of measurement

R1, *R2* are the dimensionless numbers.

5.2.7 Experience

U.S. Navy's A7 aircraft software (Henninger [B5]).

5.2.8 Tools

Spreadsheet software.

5.3 Mean time to failure (MTTF) (982 #30) (Lyu [B12] and Musa et al. [B15])

5.3.1 Definition

$$MTTF = \frac{\sum_{i}^{n} t_i}{n}$$

where t_i is the ith time between failures, ignoring repair times, for $i \geq 1$, and n is the number of times between failures.

5.3.2 Variables

t_i is the ith time between failures of a release or module.

5.3.3 Parameters

n is the number of times between failures.

5.3.4 Application

Estimate the expected time to failure (i.e., time between failures, ignoring repair times) of a release or module.

5.3.5 Data requirements

Times when failures occur in a release or module.

5.3.6 Units of measurement

t_i are the seconds, minutes, hours, and so on, as appropriate.

5.3.7 Experience

Space Shuttle Primary Avionics Software Subsystem (Schneidewind [B21]).

5.3.8 Tools

Spreadsheet software.

6. New maintainability measures

6.1 Mean time to repair (MTTR) (Lyu [B12] and Musa et al. [B15])

6.1.1 Definition

$$MTTR = \frac{\sum_{i}^{n} R_i}{n}$$

where R_i is the ith repair time, for $i \geq 1$, and n is the number of repairs. A repair may consist of 1) identifying and fixing the fault(s) that caused the system to fail, or 2) restoring the system to service without identifying or repairing the fault(s) causing the failure (e.g., rebooting the system). Repairs may be unscheduled (e.g., in response to a failure) or scheduled (as part of regular system maintenance).

6.1.2 Variables

R_i is the ith repair time of a release or module.

6.1.3 Parameters

n is the number of repair times.

6.1.4 Application

Estimate the time that will be required to repair a fault after a failure has occurred in a release or module.

6.1.5 Data requirements

R_i is the repair time.

6.1.6 Units of measurement

R_i are the seconds, minutes, hours, and so on, as appropriate.

6.1.7 Experience

Large multi-release telecommunications switching system (Lyu [B12]).

6.1.8 Tools

Spreadsheet software.

6.2 Network maintainability (Schneidewind [B19])

6.2.1 Definition

MN is the maintainability of a network (probability of maintenance required on a network):

$$MN = (MC) \times (MS)$$

6.2.2 Variables

MC is the maintainability of clients (probability of maintenance required on nc clients during time *T*).

$$MC = \frac{\sum_{i=1}^{nc} mc_i}{T}$$

MS is the maintainability of servers (probability of maintenance required on ns servers during time *T*).

$$MS = \frac{\sum_{i=1}^{ns} ms_i}{T}$$

mc_i is the maintenance time on client *i* during time *T*.

ms_i is the maintenance time on server *i* during time *T*.

6.2.3 Parameters

T is the scheduled operational time.

nc is the number of client nodes in the network.

ns is the number of server nodes in the network.

6.2.4 Application

Maintainability assessment of networks.

6.2.5 Data requirements

mc_i is the maintenance time on client *i* during time *T*.

ms_i is the maintenance time on server *i* during time *T*.

T is the scheduled operational time.

6.2.6 Units of measurement

mc_i, ms_i, *T* are the seconds, minutes, hours, and days, as appropriate.

nc, *ns* are the dimensionless numbers.

6.2.7 Experience

U.S. Marine Corps Tactical Systems Support Activity for distributed system software reliability assessment and prediction (Schneidewind [B19]).

6.2.8 Tools

Spreadsheet software.

7. New availability measures

7.1 Availability (Lyu [B12] and Musa et al. [B15])

7.1.1 Definition

Availability is the probability that software is available when needed.

$$Availability = \frac{MTTF}{MTTF + MTTR}$$

7.1.2 Variables

MTTF and *MTTR*.

7.1.3 Parameters

None.

7.1.4 Application

Compute the probability that software (e.g., release or module) is available when needed.

7.1.5 Data requirements

MTTF (time to failure measured from end of repair to next failure; see 5.3) and *MTTR* (see 6.1).

7.1.6 Units of measurement

MTTF and *MTTR* are the seconds, minutes, hours, and days, as appropriate.

7.1.7 Experience

Large multi-release telecommunications switching system (Lyu [B12]).

7.1.8 Tools

Spreadsheet software.

7.2 Network availability (Schneidewind [B19])

7.2.1 Definition

A is the network availability (probability that the network is available when needed).

$$A = \frac{MTTF_n}{MTTF_n + MTTR_n} = \frac{MTBF_n - MTTR_n}{MTBF_n}$$

7.2.2 Variables

7.2.2.1 Mean time between failures

MTBF$_n$ is the MTBF for a network.

$$MTBF_n = MTBF_c \times \left(\frac{nc}{nc + ns}\right) + MTBF_s \times \left(\frac{ns}{nc + ns}\right)$$

(weighted average of $MTBF_c$ and $MTBF_s$).

MTBF$_c$ is the MTBF for clients.

$$MTBF_c = \frac{T}{FC}$$

FC is the total number of client failures during time *T*.

$$FC = \sum_{i=1}^{nc} fc_1$$

fc_i is the number of failures of client *i* during time *T*.

nc is the number of client nodes in a network.

MTBF$_s$ is the MTBF for servers.

$$MTBF_s = \frac{T}{FS}$$

FS is the total number of server failures during time *T*.

$$FS = \sum_{i=1}^{ns} fs_1$$

fs_i is the number of failures of server *i* during time *T*.

ns is the number of server nodes in a network.

7.2.2.2 Mean time to repair

MTTR$_n$ is the MTTR for a network.

$$MTTR_n = MTTR_c \times \left(\frac{nc}{nc + ns} \right) + MTTR_s \times \left(\frac{ns}{nc + ns} \right)$$

(weighted average of $MTTR_c$ and $MTTR_s$).

MTTR$_c$ is the MTTR for clients.

$$MTTR_c = \frac{\sum_{i=1}^{nc} mc_i}{FC}$$

mc_i is the maintenance time on client i during time T.

MTTR$_s$ is the MTTR for servers.

$$MTTR_s = \frac{\sum_{i=1}^{ns} ms_i}{FS}$$

ms_i is the maintenance time on server i during time T.

7.2.3 Parameters

nc is the number of client nodes in a network.

ns is the number of server nodes in a network.

T is the scheduled operational time.

7.2.4 Application

Availability assessment of networks.

7.2.5 Data requirements

fc_i is the number of failures of client i during time T.

fs_i is the number of failures of server i during time T.

mc_i is the maintenance time on client i during time T.

ms_i is the maintenance time on server i during time T.

T is the scheduled operational time.

7.2.6 Units of measurement

fc_i, fs_i are the failure counts.

mc_i, ms_i, T are the seconds, minutes, hours, and days, as appropriate.

nc, ns are the dimensionless numbers.

7.2.7 Experience

U.S. Marine Corps Tactical Systems Support Activity for distributed system software reliability assessment and prediction (Schneidewind [B19]).

7.2.8 Tools

Spreadsheet software.

Annex A

(informative)

Analysis of measures in IEEE Software Engineering Collection, IEEE Std 982.1-2006, Standard Dictionary of Measures to Produce Reliable Software

This annex, in Table A.1, provides the justification for modifying, retaining, or deleting existing measures.

The criteria that were used in deciding whether to modify, retain, or delete a measure are listed as follows. The same criteria were applied to new measures. To be eligible for inclusion, a measure should satisfy one or more of the following criteria:

a) A measure should have some minimum number of recognized uses

b) A measure should have demonstrated or potential utility in producing reliable, maintainable, and available software

c) A measure should not be overly complex, difficult to understand, difficult to implement, or require expensive tools to implement

d) A measure should be development paradigm independent (i.e., have wide applicability)

Object-oriented measures do not satisfy criteria 1 and 3. With respect to criterion 1, there is not a minimum number of recognized industry uses or applications. Although object-oriented measures have potential utility in producing reliable, maintainable, and available software, they are not sufficiently mature to include in this standard at this time.

Table A.1—Justification for modifying, retaining, or deleting measures in previous version

Analysis of existing measures	Modify	Retain	Delete	Rationale
4.1 Fault density		X		Substantial justification and good documentation presented for this measure.
4.2 Defect density	X			Generalize and simplify. Use the Space Shuttle Discrepancy Report approach.
4.3 Cumulative failure profile			X	The accumulation of failures to an asymptote does not necessarily mean that high reliability has been achieved. Better measures are predictions of remaining failures and time to failure. The remaining failures could be disastrous.
4.4 Fault-days number			X	This unnormalized measure is not meaningful because the sum of fault days is not comparable among systems. Need to divide by number of faults, but thus calculation would yield MTBF.
4.5 Functional or modular test coverage	X			This metric assumes a one-to-one relationship between test cases and requirements. Typically, there are many test cases per requirement; hence, the metric does not work as defined. It should be modified to show test cases per requirement passed.
4.6 Cause and effect graphing			X	Ambiguous. Difficult to interpret. Low usage.
4.7 Requirements traceability		X		This measure aids in identifying requirements that are either missing from or in addition to the original requirements.
4.8 Defect indices			X	Too subjective; data difficult to collect.

4.9 Error distributions			X	Use IEEE 1044 fault classification standard [B9].
4.10 Software maturity index			X	This is not a measure of maturity. It is a measure of module change rate but not a good one. The value could be negative.
4.11 Manhours per major defect detected			X	No convincing argument on interpretation. Large value could be due to critical module inspection or poor efficiency in inspection. Low value could be due to a few resources applied or efficient inspection.
4.12 Number of conflicting requirements			X	Should map from requirements to specifications. One could have multiple requirements associated with a single specification and vice versa. Example is incorrect.
4.13 Number of entries and exits per module			X	All that is needed is a rule about single entry and exit.
4.14 Software science measures			X	n_1, n_2, N_1, and N_2 proved useful on the Space Shuttle as quality discriminants. However, these measures are difficult to implement because thresholds that distinguish low from high quality must be estimated statistically. This process is beyond the skills of most practitioners. The remaining measures are very controversial.
4.15 Graph-theoretic complexity for architecture			X	Some components of the measures (e.g., d_k) are highly subjective and depend on judgments by the user. It would be infeasible to collect the data necessary to implement the measures.
4.16 Cyclomatic complexity			X	There is no scientific evidence that threshold values that must be used with this measure, such as 10, have general applicability. The number 10, as suggested by McCabe [B13], applied to software developed in the National Security Agency. Also, McCabe never suggested that his measure be used for reliability, maintainability, and availability. Rather, he was applying his measure to formulating test strategies.
4.17 Minimal unit test case determination			X	A definition is needed for an independent path. Here is one: The concept of independent path comes from graph theory. Independent paths (i.e., fundamental circuits in graph theory) have the property that no path in the fundamental circuit matrix can be obtained by a linear combination of other paths in the matrix (Schneidewind [B18]). Also, it has been shown that confining testing to independent paths does not necessarily cover the minimal number of test cases in a given program (Evangelist [B4]).
4.18 Run reliability			X	“Correct results” is not defined. In order for Run Reliability $R_k = P_r^k$, independent runs must be assumed. This assumption is unrealistic. If P_r is assumed uniformly distributed, the assumption is unrealistic. If P_r is assumed *not* to be uniformly distributed, the data to support this assumption would be very difficult to obtain. Today, the objectives for using this measure would be obtained by using the Operational Profile.
4.19 Design structure			X	This measure adds non-commensurate quantities (i.e., apples and oranges). The resultant quantity is meaningless. Also, some primitives required in the computation would be difficult to collect.
4.20 Mean time to discover the next *K* faults			X	The predicted time to next *K* faults (failures) is much more useful than the mean time. This new

				measure is proposed to replace the current measure. It is shown as #3.1 under “New Measures” (Schneidewind [B20]). The equation has been used on the following projects: Space Shuttle flight software, Naval Surface Warfare Center Trident software, U.S. Navy Program Executive Office for Theater Combatants Software Dependability Handbook, and Marine Corps Tactical System Support Activity distributed systems. Similar equations have been applied to AT&T switching systems and widely applied elsewhere (Musa [B14] and Musa et al. [B15]).
4.21 Software purity level			X	Quoting from the Consideration’s section of the measure’s description: “The application of this measure is especially applicable for those classes of models in which the total number of remaining faults in the program cannot be predicted (e.g., Moranda’s geometric model). Reliability prediction should not have to depend on such constraints. Furthermore, this measure would not be easy for the practitioner to interpret. Lastly, there are more meaningful predictors of reliability such as Time to Next Failure (e.g., New Measure #3.1) and Remaining Failures (e.g., New Measure #3.3)” (ANSI R-013-1992 [B1], Musa [B14], Musa et al. [B15], and Schneidewind [B20]).
4.22 Estimated number of faults remaining (by seeding)			X	Quoting from IEEE Std 982: “Before seeding, a fault analysis is needed to determine the types of faults and their relative frequency of occurrence expected under a particular set of software development conditions.” If a fault analysis has already been conducted, what is the point of doing the seeding?
4.23 Requirements compliance	X			This measure retains the basic concept of identifying and computing requirements deficiencies, but it updates the tool used for automating the detection of deficiencies.
4.24 Test coverage			X	This measure is similar to 4.5, Functional or modular test coverage. The latter has been used.
4.25 Data or information flow complexity			X	This measure has a hodgepodge of unrelated terms that make no sense. Just multiplying Fan In * Fan Out has this effect. Squaring this term exacerbates the problem. Weighting with *length* confounds the measure even more. Also, the procedure and data flow methodologies could be considered out of date.
4.26 Reliability growth function			X	Least squares is not a good reliability estimation technique. $R(k) = (R(u) - A/K)$ does not make sense. Why is a reliability growth parameter divided by K, number of stages? $R(k)$ and $R(u)$ are never defined. Could use predicted remaining faults instead. Could achieve the objective easier with reliability growth models that have the advantage of being implemented in the software reliability tools such as those identified in Appendix A of Lyu [B12]. No units of computation given.
4.27 Residual fault count			X	No computation given, only primitives. “Software integrity” not defined. The statement: “Ideally, the software failure rate is constant between failures” makes no sense. For reliability growth, want the

				failure rate to decrease. Assumes using NHPP will solve the problem of inter-failure time dependencies.
4.28 Failure analysis using elapsed time			X	No computation given, only one primitive! Achieving good fit with historical data does not mean model will predict well in the future.
4.29 Testing sufficiency			X	This measure is similar to 4.5, Functional or modular test coverage. The latter has been used.
4.30 Mean-time-to-failure		X		Substantial justification and good documentation presented for this measure.
4.31 Failure rate	X			Instead of the theoretical failure rate given in the measure, use the empirical failure rate computed from (Incremental Number of Failures)/(Incremental Test or Operational Time). This measure would be much easier for the practitioner to compute and apply.
4.32 Software documentation and source listings			X	A measure should not rely on the use of a questionnaire. The measure should stand on its own merits. Math models should not be a subcharacteristic of the measure.
4.33 Rely (required software reliability)			X	This is not a reliability measure. It is set of reliability attributes (e.g., severity levels).
4.34 Software release readiness			X	This measure is a hodgepodge of undefined and vague primitives like "software operator interface." The implementation of the measure is so impractical that it cannot be computed in the real world.
4.35 Completeness			X	Covered by Requirements Compliance (982 #23) (modified existing measure). Existing measure is very complex to implement.
4.36 Test accuracy			X	This measure is much too complex for practical application. Much of the data needed for its computation cannot be collected. The test coverage measure Gamma is not defined.
4.37 System performance reliability			X	This measure is beyond the scope of the standard; it is not a software reliability measure. It is a system performance measure, a function of hardware and software.
4.38 Independent process reliability			X	This measure is much too complex for practical application. Much of the data needed for its computation cannot be collected. Assumes programs have "no loops and no local dependencies exist." Assumes that a large program is composed of "logically independent modules which can be designed, implemented and tested independently."
4.39 Combined hardware and software (system) operational availability			X	As stated in the Experience section of the standard: "Few hardware/software models have been developed that can be applied to the measurement of system operational availability." In addition, the equations for this measure are very complex and difficult to use, and the data to support the measure would be difficult to obtain.

Annex B

(informative)

Software Reliability Engineering Case Study

(Keller and Schneidewind [B11])

B.1 General

In broad terms, implementing a software reliability program is a two-phased process. It consists of (1) identifying the reliability goals and (2) testing the software to see whether it conforms to the goals. The reliability goals can be ideal (e.g., zero defects) but should have some basis in reality based on tradeoffs between reliability and cost. The testing phase is more complex because it involves collecting raw defect data and using it for assessment and prediction.

The goals of the American National Standards Institute/American Institute of Aeronautics and Astronautics Recommended Practice on Software Reliability, R-013-1992 [B1] are as follows: Provide a common basis for discussion among individuals within a project and across projects; remind practitioners about the many aspects of the software reliability process that are important to consider; and advise them on how to achieve good results and avoid bad practices. Next, are described the steps on how to implement a Software Reliability Engineering (SRE) program, using the Space Shuttle as an example, and keying the appropriate measure to the recommended practice.

B.2 Implementing a Software Reliability Engineering program

The following are major steps in SRE (not necessarily in chronological order):

a) **State the safety criteria.** This might be stated, for example, as “no failure that would result in loss of life or mission.”

b) **Collect fault and failure data.** For each system, there should be a brief description of its purpose and functions and the fault and failure data, as shown below. Days # could be hours, minutes, as appropriate. Code the Problem Report Identification to indicate Software (S) failure, Hardware (H) failure, or People (P) failure.

   1) System Identification
   2) Purpose
   3) Functions
   4) Days # (since start of test)
   5) Problem Report Identification
   6) Problem Severity
   7) Failure Date
   8) Module with Fault
   9) Description of Problem

c) **Establish problem severity levels.** Use a problem severity classification, such as the following:

   1) Loss of life, loss of mission, abort mission
   2) Degradation in performance
   3) Operator annoyance

4) System OK, but documentation in error

5) Error in classifying a problem (i.e., no problem existed in the first place)

NOTE—Not all problems result in failures.[4]

d) **Implement the safety criteria.** Two criteria for software reliability levels will be defined. Then these criteria will be applied to the risk analysis of safety critical software. In the case of the Shuttle example, the "risk" will represent the degree to which the occurrence of failures does not meet required reliability levels. Three prediction quantities, remaining failures (3.4), time to next failure (3.2), and total test time to achieve specified remaining failures (3.5), are applied in item 1) and item 2) below. This is followed by a risk assessment based on the degree to which the predictions satisfy the risk criteria. Finally, guidance is provided on how to interpret reliability predictions.

   If the safety goal is the reduction of failures of a specified severity to an acceptable level of risk (Lyu [B12]), then for software to be ready to deploy, after having been tested for total time t_t, it would satisfy the following criteria:

   1) Predicted remaining failures $r(t_t) < r_c$, where r_c is a specified critical value.

   2) Predicted time to next failure $T_F(t_t) > t_m$, where t_m is mission duration.

   The predicted value of $r(t_t)$ would be obtained in accordance with section *3.3 Remaining Failures* of Schneidewind [B20].

   The predicted value of $T_F(t_t)$ would be obtained in accordance with section *3.1 Time to Next Failure(s)* of Schneidewind [B20].

   For systems that are tested and operated continuously like the Shuttle, t_t, $T_F(t_t)$, and t_m are measured in execution time. Note that, as with any methodology for assuring software safety, there is no guarantee that the expected level will be achieved. Rather, with these criteria, the objective is to reduce the risk of deploying the software to a "desired" level.

   Apply the remaining failures criterion. Criterion 1) sets the threshold on remaining failures that would be satisfied to deploy the software (i.e., no more than a specified number of failures).

   If we predict $r(t_t) \geq r_c$, we would continue to test for a total time $t_t' > t_t$ that is predicted to achieve $r(t_t') < r_c$, using the assumption that more failures will be experienced and more faults will be corrected so that the remaining failures will be reduced by the quantity $r(t_t) - r(t_t')$. If the developer does not have the resources to satisfy the criterion or cannot satisfy the criterion through additional testing, the risk of deploying the software prematurely should be assessed (see the next section). It is known that it is impossible to demonstrate the absence of faults (Dijkstra [B3]); however, the risk of failures occurring can be reduced to an acceptable level, as represented by r_c. This scenario is shown in Figure B.1. In case A, $r(t_t) < r_c$ is predicted and the mission begins at t_t. In case B, $r(t_t) \geq r_c$ is predicted and the mission would be postponed until the software is tested for total time t_t' when $r(t_t') < r_c$ is predicted. In both cases, criterion 2) would also be required for the mission to begin.

[4] Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

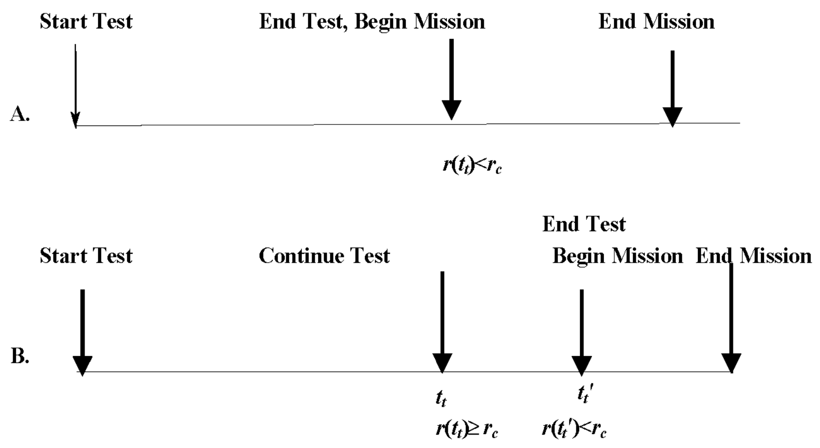

Figure B.1—Remaining failures criterion scenario

Apply the time to next failure criterion. Criterion 2 specifies that the software needs to survive for a time greater than the duration of the mission. If $T_F(t_t) \leq t_m$ is predicted, the software is tested for a total time $t_t'' > t_t$ that is predicted to achieve $T_F(t_t'') > t_m$, using the assumption that more failures will be experienced and faults corrected so that the time to next failure will be increased by the quantity $T_F(t_t'') - T_F(t_t)$. Again, if it is infeasible for the developer to satisfy the criterion for lack of resources or failure to achieve test objectives, the risk of deploying the software prematurely should be assessed (see the next section). This scenario is shown in Figure B.2. In case A, $T_F(t_t) > t_m$ is predicted and the mission begins at t_t. In case B, $T_F(t_t) \leq t_m$ is predicted, and in this case, the mission would be postponed until the software is tested for total time t_t'' when $T_F(t_t'') > t_m$ is predicted. In both cases, criterion 1) would also be required for the mission to begin. If neither criterion is satisfied, the software is tested for a time that is the greater of t_t' or t_t''.

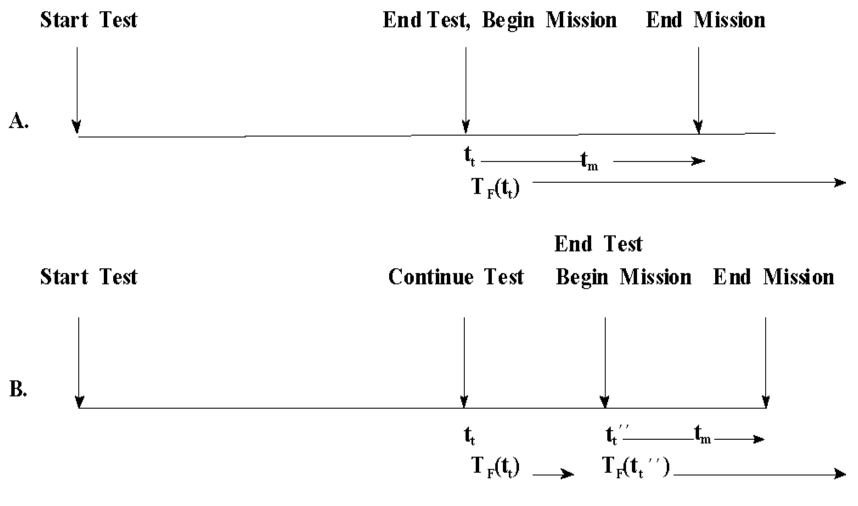

Figure B.2—Time to Next Failure criterion scenario

e) **Make a risk assessment.** Safety Risk pertains to executing the software of a safety critical system where there is the chance of injury (e.g., astronaut injury or fatality), damage (e.g., destruction of the Shuttle), or loss (e.g., loss of the mission), if a serious software failure occurs during a mission.

The amount of total test time t_t can be considered a measure of the degree to which software reliability goals have been achieved. This is particularly the case for systems like the Space Shuttle where the software is subjected to continuous and rigorous testing for several years in multiple facilities, using a variety of operational and training scenarios (e.g., by the contractor in Houston, by NASA in Houston for astronaut training, and by NASA at Cape Canaveral). Total test time t_t can be viewed as an input to a risk reduction process and $r(t_t)$ and $T_F(t_t)$ as the outputs, with r_c and t_m as "risk criteria levels" of reliability that control the process. Although it can be recognized that total test time is not the only consideration in developing test strategies and that there are other important factors, like the consequences for reliability and cost, in selecting test cases (Nikora et al. [B17]), nevertheless, for the foregoing reasons, total test time has been found to be strongly positively correlated with reliability growth for the Space Shuttle (Musa [B14]).

Evaluate remaining failures risk. The mean value of the risk criterion metric (RCM) for criterion 1 is formulated as in Equation (B.1):

$$RCM\ r(t_t) = \frac{(r(t_t) - r_c)}{r_c} = \left(\frac{r(t_t)}{r_c}\right) - 1 \tag{B.1}$$

Equation (B.1) is plotted in Figure B.3 as a function of t_t for $r_c = 1$, for OID, where positive, zero, and negative values correspond to $r(t_t) > r_c$, $r(t_t) = r_c$, and $r(t_t) < r_c$, respectively. In Figure B.3, these values correspond to the following regions: CRITICAL (i.e., above the X-axis, predicted remaining failures are greater than the specified value); NEUTRAL (i.e., on the X-axis, predicted remaining failures are equal to the specified value); and DESIRED (i.e., below the X-axis, predicted remaining failures are less than the specified value, which could represent a "safe" threshold or, in the Shuttle example, an "error-free" condition boundary). This graph is for the Shuttle Operational Increment OID (with many years of shelf life): a software system comprising modules and configured from a series of builds to meet Shuttle mission functional requirements. In this example it can be seen that at approximately $t_t = 57$, the risk transitions from the CRITICAL region to the DESIRED region.

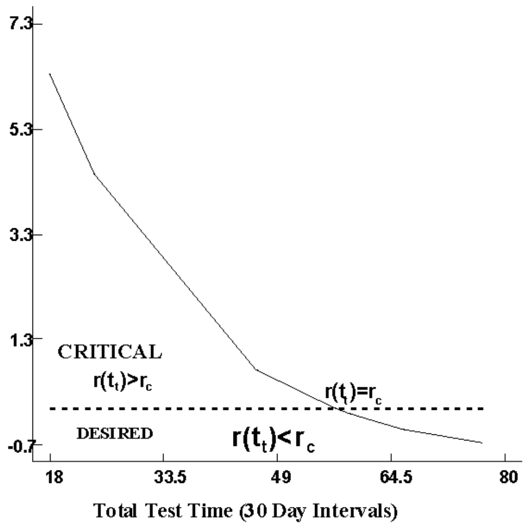

Figure B.3—RCM for Remaining Failures (*r* =1), OID

Evaluate Time to Next Failure Risk. Similarly the mean value of the risk criterion metric (RCM) for criterion 2 is formulated as in Equation (B.2):

$$RCM\ T_F(t_t) = \frac{(t_m - T_F(t_t))}{t_m} = \left(\frac{T_F(t_t)}{t_m}\right) - 1 \tag{B.2}$$

Equation (B.2) is plotted in Figure B.4 as a function of t_t for t_m = 8 days (a typical mission duration), for OIC, where positive, zero, and negative risk corresponds to $T_F(t_t) < t_m$, $T_F(t_t) = t_m$, and $T_F(t_t) > t_m$, respectively. In Figure B.4, these values correspond to the following regions: CRITICAL (i.e., above the X-axis, predicted time to next failure is less than the specified value); NEUTRAL (i.e., on the X-axis, predicted time to next failure is equal to the specified value); and DESIRED (i.e., below the X-axis, predicted time to next failure is greater than the specified value). This graph is for the Shuttle operational increment OIC. In this example, the RCM is in the DESIRED region at all values of t_t.

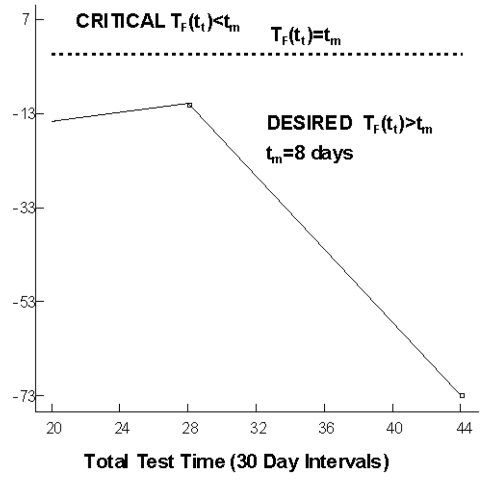

Figure B.4—RCM for Time to Next Failure (t_m = 8 days), OIC

f) **Interpret Software Reliability Predictions:** Successful use of statistical modeling in predicting the reliability of a software system requires a thorough understanding of precisely how the resulting predictions are to be interpreted and applied (Musa [B14]). The Shuttle software (430 KLOC) is frequently modified, at the request of NASA, to add or change capabilities using a constantly improving process. Each of these successive versions constitutes an upgrade to the preceding software version. Each new version of the software (designated as an Operational Increment, OI) contains software code that has been carried forward from each of the previous versions ("previous-version subset") as well as new code generated for that new version ("new-version subset"). We have found that by applying a reliability model independently to the code subsets we can obtain satisfactory composite predictions for the total version.

It is essential to recognize that this approach requires an accurate code change history so that every failure can be uniquely attributed to the version in which the defective line(s) of code were first introduced. In this way, it is possible to build a separate failure history for the new code in each release. To apply SRE to your software system, you should consider breaking your systems and processes down into smaller elements to which a reliability model can be more accurately applied. Using this approach, we have been successful in applying SRE to predict the reliability of the Shuttle software for NASA.

Annex C

(informative)

Glossary

For the purposes of this document, the following terms and definitions apply. These and other terms within IEEE standards are found in *The Authoritative Dictionary of IEEE Standards Terms* [B8].[5]

availability: The degree to which a system or component is operational and accessible when required for use. Often expressed as a probability. (adapted from IEEE Std 610.12™-1990 [B10])

failure: The inability of a system or component to perform its required functions within specified performance requirements. (adapted from IEEE Std 610.12-1990 [B10])

fault: An incorrect step, process, or data definition in a computer program. (adapted from IEEE Std 610.12-1990 [B10])

reliability: The ability of a system or component to perform its required functions under stated conditions for a specified period of time. (adapted from IEEE Std 610.12-1990 [B10])

[5] IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (http://standards.ieee.org/).

Annex D

(informative)

Bibliography

[B1] ANSI/AIAA R-013-1992, Recommended Practice for Software Reliability.[6]

[B2] Binder, R. V., *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Reading, MA: Addison-Wesley, 2000.

[B3] Dijkstra, E., "Structured programming," In: Buxton, J.N., and Randell, B., eds. *Software Engineering Techniques*. Brussels Belgium: NATO Scientific Affairs Division, April 1970, pp. 84–88.

[B4] Evangislist, W. M., "Software complexity metric sensitivity to program restructuring rules," *Journal of Systems and Software*, vol. 3, pp. 231–243, 1983.

[B5] Fenton, N. E. and Pfleeger, S. L., *Software Metrics: A Rigorous & Practical Approach*, 2d ed. Boston, MA: PWS Publishing Company, 1997.

[B6] Fischer, K. F. and Walker, M. G., "Improved software reliability through requirement verification," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 233–239, August 1979.

[B7] Henninger, K., "Specifying software requirements for complex systems, new techniques and their application," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 1, pp. 1–14, Jan. 1980.

[B8] IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms,* Seventh Edition, New York, Institute of Electrical and Electronics Engineers, Inc.[7]

[B9] IEEE Std 1044™-1993, IEEE Standard Classification for Software Anomalies.[8]

[B10] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

[B11] Keller, T. and Schneidewind, N. F., "A successful application of software reliability engineering for the NASA Space Shuttle," *Software Reliability Engineering Case Studies, International Symposium on Software Reliability Engineering*, Albuquerque, NM, pp. 71–82, Nov. 4, 1997.

[B12] Lyu, M. R., *Handbook of Software Reliability Engineering*. New York: IEEE Computer Society Press and McGraw-Hill, 1996.

[B13] McCabe, T. J., "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[B14] Musa, J. D., *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*. New York: McGraw-Hill, 1999.

[6] ANSI publications are available from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

[7] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).

[8] The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

[B15] Musa, J. D., et al., *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill, 1987.

[B16] Nikora, A. and Munson, J., "Determining fault insertion rates for evolving software systems," *Proceedings of the Ninth International Symposium on Software Reliability Engineering*, Paderborn, Germany, pp. 306–315, Nov. 4–7, 1998.

[B17] Nikora, A., Schneidewind, N., and Munson, J., "IV&V issues in achieving high reliability and safety in critical control software," Final Report, Volume 1—Measuring and Evaluating the Software Maintenance Process and Metrics-Based Software Quality Control, Volume 2—Measuring Defect Insertion Rates and Risk of Exposure to Residual Defects in Evolving Software Systems, and Volume 3—Appendices, Jet Propulsion Laboratory, National Aeronautics and Space Administration, Pasadena, CA, Jan 19, 1998.

[B18] Schneidewind, N. F., "Application of program graphs and complexity analysis to software development and testing," *IEEE Transactions on Reliability*, vol. R-28, no.3, pp. 192–198, Aug. 1979.

[B19] Schneidewind, N. F., "Software reliability engineering for client-server systems," *Proceedings of The Seventh International Symposium on Software Reliability Engineering*, White Plains, NY, pp. 226–235, Oct. 30–Nov. 2, 1996.

[B20] Schneidewind, N. F., "Reliability modeling for safety critical software," *IEEE Transactions on Reliability*, vol. 46, no.1, pp.88–98, Mar. 1997.

[B21] Schneidewind, N. F., "Measuring and evaluating maintenance process using reliability, risk, and test metrics," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, pp. 768–781, Nov./Dec. 1999.

[B22] Schneidewind, N. F., "Investigation of the risk to software reliability and maintainability of requirements changes," *Proceedings of the International Conference on Software Maintenance*, Florence, Italy, pp. 127–136, Nov. 7–9, 2001.

[B23] *Webster's New Collegiate Dictionary.* Springfield, MA: Merriam-Webster, Inc.