



IEEE Recommended Practice on Software Reliability

IEEE Reliability Society

Sponsored by the
Standards Committee

1633TM

IEEE
3 Park Avenue
New York, NY 10016-5997, USA
27 June 2008

IEEE Std 1633TM-2008

IEEE Recommended Practice on Software Reliability

Sponsor

**Standards Committee
of the
IEEE Reliability Society**

Approved 27 March 2008

IEEE-SA Standards Board

© 1999 IEEE. Figure F.6 is reprinted with permission from the IEEE and Samuel Keene (author), from his paper presented at the *Tenth International Symposium on Software Reliability Engineering*.

Abstract: The methods for assessing and predicting the reliability of software, based on a life-cycle approach to software reliability engineering, are prescribed in this recommended practice. It provides information necessary for the application of software reliability (SR) measurement to a project, lays a foundation for building consistent methods, and establishes the basic principle for collecting the data needed to assess and predict the reliability of software. The recommended practice prescribes how any user can participate in SR assessments and predictions.

Keywords: software reliability

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

Copyright © 2008 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 27 June 2008. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

PDF: ISBN 978-0-7381-5406-0 STD95776
Print: ISBN 978-0-7381-5407-7 STDPD95776

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 IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied “**AS IS**.”

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.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon his or her independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

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 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 societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal interpretation of the IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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. Comments on standards and requests for interpretations should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

Authorization to photocopy portions of any individual standard for internal or personal use is granted by The Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Introduction

This introduction is not part of IEEE Std 1633-2008, IEEE Recommended Practice on Software Reliability.

Software reliability engineering (SRE) is an established discipline that can help organizations improve the reliability of their products and processes. The American Institute of Aeronautics and Astronautics (AIAA) defines SRE as “the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems.” This recommended practice is a composite of models and tools and describes the “what and how” of SRE. It is important for an organization to have a disciplined process if it is to produce high reliability software. The process is described and enhanced to include a life-cycle approach to SRE that takes into account the risk to reliability due to requirements changes. A requirements change may induce ambiguity and uncertainty in the development process that cause errors in implementing the changes. Subsequently, these errors may propagate through later phases of development and maintenance. These errors may result in significant risks associated with implementing the requirements. For example, reliability risk (i.e., risk of faults and failures induced by changes in requirements) may be incurred by deficiencies in the process (e.g., lack of precision in requirements). Figure a shows the overall SRE closed-loop holistic process. The arrows signify the flow of software product, information, and/or failure data.

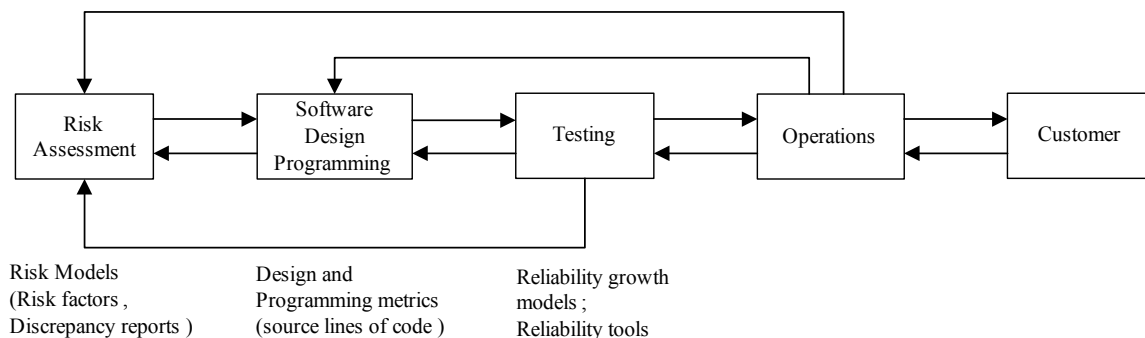


Figure a—SRE process

The scope of this recommended practice is to address software reliability (SR) methodologies and tools. This recommended practice does not address systems reliability, software safety, nor software security. The recommended practice only briefly addresses software quality. This recommended practice provides a common baseline for discussion and prescribes methods for assessing and predicting the reliability of software. The recommended practice is intended to be used in support of designing, developing, and testing software and to provide a foundation on which practitioners and researchers can build consistent methods for assessing the reliability of software. It is intended to meet the needs of software practitioners and users who are confronted with varying terminology for reliability measurement and a plethora of models and data collection methods. This recommended practice contains information necessary for the application of SR measurement to a project. This includes SR activities throughout the software life cycle (SLC) starting at requirements generation by identifying the application, specifying requirements, and analyzing requirements and continuing into the implementation phases. It also serves as a reference for research on the subject of SR.

It includes guidance on the following:

- Common terminology.
- Assessment of SR risk.

- Determining whether previously applied software processes are likely to produce code that satisfies a given SR requirement.
- Software design process improvement evaluation and software quality.
- SR assessment procedures (i.e., measure current software reliability).
- Test selection and model selection.
- Data collection procedures to support SR estimation and prediction.
- Determining when to release a software system, or to stop testing the software and implement corrections.
- SR prediction (i.e., predict future software reliability). This includes the calculation of the probability of occurrence of the next failure for a software system, and other reliability metrics.
- Identify elements in a software system that are leading candidates for redesign to improve reliability.

Revisions to the document and notes

This document is a revision of AIAA R-013-1992, Recommended Practice on Software Reliability. The following changes and additions are designed to enhance its usability:

- “Recommended models” has been changed to “initial models” to reflect the fact that this document is a recommended practice. The meaning of “initial models” is that models in this category are designated for initial use. If none of these models is satisfactory for the user’s application, the models described in Annex A can be considered.
- Life-cycle approach to SRE.
- SRE process diagram (see Figure a).
- Inclusion of reliability requirements risk assessment.
- Additions to 5.1.
- Identify application.
- Specify the reliability requirement.
- Allocate the requirement.
- Correct designation of informative clauses.
- Simplified and clarified language.
- Elimination of mathematics that do not support the objectives of the recommended practice.
- Correct use of “shall,” “should,” and “may” to indicate conformance with the recommended practice.
- Correction of errors.
- Upgrade of Schneidewind initial model.
- Addition of John Musa’s latest book [B55]^a as a reference.
- Deletion of Assumption 1 in the Musa/Okumoto model.
- The Littlewood/Verrall model has been moved from “initial models” to Annex A because many of its terms were not defined.

^a The numbers in brackets correspond to those of the bibliography in Annex G.

Structure of the recommended practice

This recommended practice contains five clauses and seven annexes as follows:

- Clause 1 is the introduction, including scope, purpose, audience, and relationships between hardware and SR.
- Clause 2 contains definitions of terms used in this recommended practice.
- Clause 3 gives an overview of SRE.
- Clause 4 provides a list of activities that should be carried out in order to conform with this recommended practice.
- Clause 5 provides information on recommended SR prediction models.
- Annex A provides information on additional SR prediction models.
- Annex B describes methods for combining hardware and SR predictions into a system reliability prediction.
- Annex C provides information on using the recommended practice to help obtain system reliability (including both hardware and SR).
- Annex D contains Table D.1, which lists the two most popular SR measurement tools.
- Annex E contains a comparison of constant, linearly decreasing, and exponentially decreasing models.
- Annex F provides SR prediction tools prior to testing during the early design phases.
- Annex G contains a bibliography of over 70 papers and books about SRE.

Clause 1, Clause 2, and Clause 3 should be read by all users. Clause 3, Annex A, and Annex C provide the basis for establishing the process and the potential uses of the process. Subclause 5.6.1 provides the foundation for establishing an SR data collection program, as well as what information needs to be collected to support the recommended models, which is also described in Annex A. Annex D identifies tools that support the reliability database, as well as the recommended models and the analysis techniques described in Clause 4, Annex A, and Annex C. Finally, to improve the state of the art in SRE continuously, recommended practice users should typically review Clause 1, Clause 2, and Clause 3 and begin applying the techniques described in 5.5, and concluding with Annex D and Annex F on reliability tools.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association Web site at <http://ieeexplore.ieee.org/xpl/standards.jsp>, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA Web site at <http://standards.ieee.org>.

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 recommended practice may require use of subject matter covered by patent rights. By publication of this recommended practice, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this recommended practice are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this recommended practice was submitted to the IEEE-SA Standards Board for approval, the Software Reliability Working Group had the following membership:

Norman F. Schneidewind, *Chair*
Louis J. Gullo, *Sponsor Chair and Final Editor*
Kadir Alpaslan Demir, *Editor*

William Everett
William Farr
Herbert Hecht
Michael Hinchey
Samuel Keene

Theodore Keller
Michael Lyu
John Musa
Allen Nikora
Martin Shooman

George Stark
Jeff Voas
Mladen Vouk
Dolores Wallace
Linda Wilbanks

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

Ahmed Abdelhalim
William J. Ackerman
S. Aggarwal
Ali Al Awazi
Bakul Banerjee
Charles Barest
Hugh Barrass
Thomas Basso
Steven Bezner
Gregory M. Bowen
Juan Carreon
Lanna Carruthers
Norbert N. Carte
Lawrence Catchpole
Weijen Chen
Keith Chow
Raul Colcher
Tommy Cooper
Paul Croll
Geoffrey Darnton
Matthew Davis
Kadir Alpaslan Demir
Bostjan K. Derganc
Antonio Doria
Neal Dowling
Scott P. Duncan
Sourav Dutta
Kameshwar Eranki
Carla Ewart
Harriet Feldman
John Fendrich
Andrew Fieldsend
Kenneth Fodero
Andre Fournier
Avraham Freedman
Michael Geipel
Gregg Giesler
Allan Gillard
Colin M. Glanville
H. Glickenstein

John Garth Glynn
Ron Greenthaler
Randall Groves
C. Guy
Ajit Gwal
John Harauz
Mark Henley
Rutger A. Heunks
Werner Hoelzl
Robert Holibaugh
Chi T. Hon
Peter Hung
Mark Jaeger
James Jones
Lars Juhlin
Piotr Karocki
Samuel Keene
Robert B. Kelsey
Mark Knight
Thomas Kurihara
George Kyle
Marc Lacroix
Daniel Lindberg
G. Luri
L. J. Tajerian Martinez
Richard A. McBride
Michael McCaffrey
Jonathon McLendon
Earl Meiers
Gary Michel
William Milam
James Moore
Thomas Mullins
Jerry Murphy
Rajesh Murthy
John Musa
Michael S. Newman
Mark Paulk
Miroslav Pavlovic
Robert Peterson
Peter Petrov

Ulrich Pohl
Iulian Profir
Annette Reilly
Michael Roberts
Robert Robinson
Terence Rout
Michael Rush
Randall Safier
James Sanders
Helmut H. Sandmayr
Bartien Sayogo
Robert Schaaf
Hans Schaefer
Norman F. Schneidewind
David J. Schultz
Stephen Schwarm
Lynn Simms
Carl Singer
James Sivak
David Smith
Steven Smith
Luca Spotorno
Manikantan Srinivasan
Thomas Starai
James St.Clair
Walter Struppler
Gerald Stueve
Mark Sturza
Noriaki Takenoue
Joseph Tardo
John Thywissen
Thomas Tullia
Vincent J. Tume
Mladen Vouk
John Walz
Robert Wilkens
David Willow
Don Wright
Oren Yuen
Janusz Zalewsk

When the IEEE-SA Standards Board approved this recommended practice on 27 March 2008, it had the following membership:

Robert M. Grow, *Chair*
Thomas Prevost, *Vice Chair*
Steve M. Mills, *Past Chair*
Judith Gorman, *Secretary*

Victor Berman
Richard DeBlasio
Andy Drozd
Mark Epstein
Alexander Gelman
William Goldbach
Arnie Greenspan
Ken Hanus

Jim Hughes
Richard Hulett
Young Kyun Kim
Joseph L. Koepfinger*
John Kulick
David J. Law
Glenn Parsons

Ron Petersen
Chuck Powers
Narayanan Ramachandran
Jon Walter Rosdahl
Anne-Marie Sahazizian
Malcolm Thaden
Howard Wolfman
Don Wright

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Michael H. Kelley, *NIST Representative*

Lisa Perry
IEEE Standards Project Editor

Matthew J. Ceglia
IEEE Standards Program Manager, Technical Program Development

Contents

1. Overview	1
1.1 Scope	1
1.2 Purpose	1
1.3 Intended audience	1
1.4 Applications of software reliability engineering.....	2
1.5 Relationship to hardware reliability.....	2
2. Definitions	3
3. Software reliability modeling—Overview, concepts, and advantages	5
3.1 General	5
3.2 Basic concepts	5
3.3 Limitations of software reliability assessment and prediction.....	6
3.4 Prediction model advantages/limitations	6
4. Software reliability assessment and prediction procedure.....	8
4.1 General	8
4.2 Software reliability procedure	8
5. Software reliability estimation models	16
5.1 Introduction	16
5.2 Criteria for model evaluation.....	17
5.3 Initial models	19
5.4 Initial model—Musa/Okumoto logarithmic Poisson execution time model.....	35
5.5 Experimental approaches.....	37
5.6 Software reliability data.....	37
Annex A (informative) Additional software reliability estimation models	41
Annex B (informative) Determining system reliability	47
Annex C (informative) Using reliability models for developing test strategies	49
Annex D (informative) Automated software reliability measurement tools.....	53
Annex E (informative) A comparison of constant, linearly decreasing, and exponentially decreasing models	54
Annex F (informative) Software reliability prediction tools prior to testing.....	61
Annex G (informative) Bibliography	69

IEEE Recommended Practice on Software Reliability

IMPORTANT NOTICE: *This standard is not intended to assure safety, security, health, or environmental protection in all circumstances. Implementers of the standard are responsible for determining appropriate safety, security, environmental, and health practices or regulatory requirements.*

This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading “Important Notice” or “Important Notices and Disclaimers Concerning IEEE Documents.” They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IPR/disclaimers.html>.

1. Overview

1.1 Scope

Software reliability (SR) models have been evaluated and ranked for their applicability to various situations. Many improvements have been made in SR modeling and prediction since 1992. This revised recommended practice reflects those advances in SR since 1992, including modeling and prediction for distributed and network systems. Situation specific usage guidance was refined and updated. The methodologies and tools included in this recommended practice are extended over the software life cycle (SLC).

1.2 Purpose

The recommended practice promotes a systems approach to SR prediction. Although there are some distinctive characteristics of aerospace software, the principles of reliability are generic, and the results can be beneficial to practitioners in any industry.

1.3 Intended audience

This recommended practice is intended for use by both practitioners (e.g., managers, software developers/engineers, software acquisition personnel, technical managers, and software quality and SR personnel) and researchers. Researchers are considered to be academics in universities and personnel doing research work in government and industry laboratories.

It is assumed that users of this recommended practice have a basic understanding of the SLC and an understanding of statistical concepts.

1.4 Applications of software reliability engineering

The techniques and methodologies presented in this recommended practice have been successfully applied to software projects by industry practitioners in order to do the following:

- Assess SR risk
- Indicate whether a previously applied software process is likely to produce code that satisfies a given SR requirement
- Provide a measure for software design process improvement evaluation and software quality
- SR assessment procedures (i.e., measure current software reliability)
- Data collection procedures to support SR estimation and prediction
- Determine when to release a software system, or to stop testing the software and make improvements
- Calculate the probability of occurrence of the next failure for a software system, and other reliability metrics
- Identify elements in a software system that are leading candidates for redesign to improve reliability
- Indicate software maintenance effort by assessing SR
- Assist software safety certification

1.5 Relationship to hardware reliability

The creation of software and hardware products is alike in many ways and can be similarly managed throughout design and development. While the management techniques may be similar, there are genuine differences between hardware and software (see Kline [B36], Lipow and Shooman [B39]¹). Examples are as follows:

- Changes to hardware require a series of important and time-consuming steps: capital equipment acquisition, component procurement, fabrication, assembly, inspection, test, and documentation. Changing software is frequently more feasible (although effects of the changes are not always clear) and often requires only testing and documentation.
- Software has no physical existence. It includes data as well as logic. Any item in a file can be a source of failure.
- One important difference is that hardware is constrained by physical law. One effect is that testing is simplified, because it is possible to conduct limited testing and use knowledge of the physics of the device to interpolate behavior that was not explicitly tested. This is not possible with software, since a minor change can cause failure.
- Software does not wear out. Furthermore, failures attributable to software faults come without advance warning and often provide no indication they have occurred. Hardware, on the other hand, often provides a period of graceful degradation.

¹ The numbers in brackets correspond to those of the bibliography in Annex G.

- Software may be more complex than hardware, although exact software copies can be produced, whereas manufacturing limitations affect hardware.
- Repair generally restores hardware to its previous state. Correction of a software fault always changes the software to a new state.
- Redundancy and fault tolerance for hardware are common practice. These concepts are only beginning to be practiced in software.
- Hardware reliability is expressed in wall clock time. SR may be expressed in execution, elapsed, or calendar time.

Despite the previously listed differences, hardware and SR must be managed as an integrated system attribute. However, these differences must be acknowledged and accommodated by the techniques applied to each of these two types of subsystems in reliability analyses.

2. Definitions

For the purposes of this recommended practice, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B22] and IEEE Std 610.12™-1990 [B23] should be referenced for terms not defined in this clause.

2.1 assessment: Determining what action to take for software that fails to meet goals (e.g., intensify inspection, intensify testing, redesign software, and revise process).

NOTE—The formulation of test strategies is also part of assessment. Test strategy formulation involves the determination of priority, duration, and completion date of testing, allocation of personnel, and allocation of computer resources to testing.²

2.2 calendar time: Chronological time, including time during which a computer may not be running.

2.3 clock time: Elapsed wall clock time from the start of program execution to the end of program execution.

2.4 error: **(A)** A discrepancy between a computed, observed, or measured value or condition and the true, specified or theoretically correct value or condition. **(B)** Human action that results in software containing a fault. A coding error may be considered a defect, but it may or may not be considered a fault. Usually, any error or defect that is detected by the system and or observed by the user is deemed a fault. All faults are serious errors that should be corrected. Not all errors are faults.

NOTE—Examples include omission or misinterpretation of user requirements in a software specification and incorrect translation or omission of a requirement in the design specification.

2.5 execution time: **(A)** The amount of actual or central processor time used in executing a program. **(B)** The period of time during which a program is executing.

2.6 failure: **(A)** The inability of a system or system component to perform a required function within specified limits. **(B)** The termination of the ability of a functional unit to perform its required function. **(C)** A departure of program operation from program requirements.

NOTE—A failure may be produced when a fault is encountered and a loss of the expected service to the user results.

² Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this recommended practice.

2.7 failure rate: (A) The ratio of the number of failures of a given category or severity to a given period of time; for example, failures per second of execution time, failures per month. Synonymous with failure intensity. (B) The ratio of the number of failures to a given unit of measure, such as failures per unit of time, failures per number of transactions, failures per number of computer runs.

2.8 failure severity: A rating system for the impact of every recognized credible software failure mode.

NOTE—The following is an example of a rating system:

- Severity #1: Loss of life or system
- Severity #2: Affects ability to complete mission objectives
- Severity #3: Workaround available, therefore minimal effects on procedures (mission objectives met)
- Severity #4: Insignificant violation of requirements or recommended practices, not visible to user in operational use
- Severity #5: Cosmetic issue that should be addressed or tracked for future action, but not necessarily a present problem.

2.9 fault: (A) A defect in the code that can be the cause of one or more failures. (B) An accidental condition that causes a functional unit to fail to perform its required function. A fault is synonymous with a bug. A fault is an error that should be fixed with a software design change.

2.10 fault tolerance: The survival attribute of a system that allows it to deliver the required service after faults have manifested themselves within the system.

2.11 integration: The process of combining software elements, hardware elements, or both into an overall system.

2.12 maximum likelihood estimation: A form of parameter estimation in which selected parameters maximize the probability that observed data could have occurred.

2.13 module: (A) A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, input to or output from an assembler, compiler, linkage editor, or executive routine. (B) A logically separable part of a program.

2.14 operational: Pertaining to the status given a software product once it has entered the operation and maintenance phase.

2.15 parameter: A variable or arbitrary constant appearing in a mathematical expression, each value of which restricts or determines the specific form of the expression.

2.16 reliability risk: The probability that requirements changes will decrease reliability.

2.17 software quality: (A) The totality of features and characteristics of a software product that bear on its ability to satisfy given needs, such as conforming to specifications. (B) The degree to which software possesses a desired combination of attributes. (C) The degree to which a customer or user perceives that software meets his or her composite expectations. (D) The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer.

2.18 software reliability (SR): (A) The probability that software will not cause the failure of a system for a specified time under specified conditions. (B) The ability of a program to perform a required function under stated conditions for a stated period of time.

NOTE—For definition (A), the probability is a function of the inputs to and use of the system, as well as a function of the existence of faults in the software. The inputs to the system determine whether existing faults, if any, are encountered.

2.19 software reliability engineering (SRE): The application of statistical techniques to data collected during system development and operation to specify, estimate, or assess the reliability of software-based systems.

2.20 software reliability estimation: The application of statistical techniques to observed failure data collected during system testing and operation to assess the reliability of the software.

2.21 software reliability model: A mathematical expression that specifies the general form of the software failure process as a function of factors such as fault introduction, fault removal, and the operational environment.

2.22 software reliability prediction: A forecast or assessment of the reliability of the software based on parameters associated with the software product and its development environment.

3. Software reliability modeling—Overview, concepts, and advantages

3.1 General

Software is a complex intellectual product. Inevitably, some errors are made during requirements formulation as well as during designing, coding, and testing the product. The development process for high-quality software includes measures that are intended to discover and correct faults resulting from these errors, including reviews, audits, screening by language-dependent tools, and several levels of test. Managing these errors involves describing the errors, classifying the severity and criticality of their effects, and modeling the effects of the remaining faults in the delivered product, and thereby working with designers to reduce their number of errors and their criticality.

NOTE—The IEEE standard for classifying errors and other anomalies is IEEE Std 1044™-1993 [B25].

Dealing with faults costs money. It also impacts development schedules and system performance (through increased use of computer resources such as memory, CPU time, and peripherals requirements). Consequently, there can be too much as well as too little effort spent dealing with faults. The system engineer (along with management) can use reliability estimation and assessment to understand the current status of the system and make tradeoff decisions.

3.2 Basic concepts

Clause 3 describes the basic concepts involved in SRE and addresses the advantages and limitations of SR prediction and estimation. The basic concept of reliability predictions and assessments of electronic systems and equipment is described in IEEE Std 1413™-1998 [B28]. The objective of IEEE Std 1413-1998 [B28] is to identify required elements for an understandable, credible reliability prediction, which will provide the users of the prediction sufficient information to evaluate the effective use of the prediction results. A reliability prediction should have sufficient information concerning inputs, assumptions, and uncertainty, such that the risk associated with using the prediction results would be understood.

There are at least two significant differences between hardware reliability and SR. First, software does not fatigue or wear out. Second, due to the accessibility of software instructions within computer memories, any line of code can contain a fault that, upon execution, is capable of producing a failure.

An SR model specifies the general form of the dependence of the failure process on the principal factors that affect it: fault introduction, fault removal, and the operational environment. The failure rate (failures per unit time) of a software system is generally decreasing due to fault identification and removal, as shown in Figure 1. At a particular time, it is possible to observe a history of the failure rate of the software. SR modeling is done to estimate the form of the curve of the failure rate by statistically estimating the parameters associated with the selected model. The purpose of this measure is twofold: 1) to estimate the extra execution time during test required to meet a specified reliability objective and 2) to identify the expected reliability of the software when the product is released.

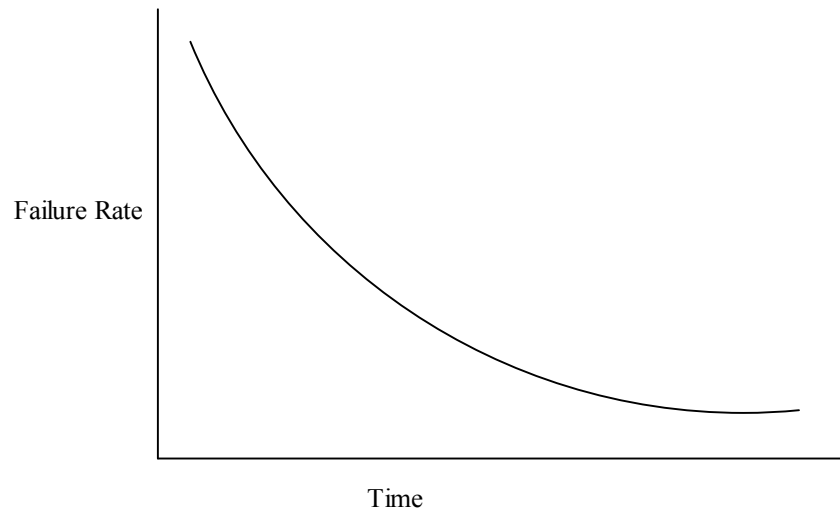


Figure 1—SR failure rate curve

3.3 Limitations of software reliability assessment and prediction

SR models can both assess and predict reliability. The former deals with measuring past and current reliabilities. The latter provides forecasts of future reliability. The word “prediction” is not intended to be used in the common dictionary sense of foretelling future events, particularly failures, but instead as an estimate of the probabilities of future events. Both assessment and prediction need good data if they are to yield good forecasts. Good data implies accuracy (that data is accurately recorded at the time the events occurred) and pertinence (that data relates to an environment that is equivalent to the environment for which the forecast is to be valid). A negative example with respect to accuracy is the restricting of failure report counts to those, which are completely filled out. This is negative because they may represent a biased sample of the total reports. A negative example with respect to pertinence would be the use of data from early test runs at an uncontrolled workload to forecast the results of a later test executed under a highly controlled workload.

3.4 Prediction model advantages/limitations

The premise of most prediction models is that the failure rate is a direct function of the number of faults in the program and that the failure rate will be reduced (reliability will be increased) as faults are detected and eliminated during test or operations. This premise is reasonable for the typical test environment and it has been shown to give credible results when correctly applied. However, the results of prediction models will be adversely affected by the following factors:

- Change in failure criteria
- Significant changes in the code under test
- Significant changes in the computing environment.

All of these factors will require the estimation of a new set of reliability model parameters. Until these can be established, the effectiveness of the model will be impaired. Estimation of new parameters depends on the measurement of several execution time intervals between failures or failure counts in intervals.

Major changes can occur with respect to several of the previously listed factors when software becomes operational. In the operational environment, the failure rate is a function of the fault content of the program, of the variability of input and computer states, and of software maintenance policies. The latter two factors are under management control and are utilized to assess an expected or desired range of values for the failure rate or the downtime due to software causes. Examples of management action that decrease the failure rate include avoidance of high workloads and avoidance of data combinations that have caused previous failures (see Gifford and Spector [B16], Iyer and Velardi [B30]). Software in the operational environment may not exhibit the reduction in failure rate with execution time that is an implicit assumption in most estimation models (see Hecht and Hecht [B19]). Knowledge of the management policies is therefore essential for selection of an SR estimation procedure for the operational environment. Thus, the prediction of operational reliability from data obtained during test may not hold true during operations.

Another limitation of SR prediction models is their use in verifying ultra-high reliability requirements. For example, if a program executes successfully for x hours, there is a 0.5 probability that it will survive the next x hours without failing (see Fenton and Littlewood [B14]). Thus, to have the kind of confidence needed to verify a 10^{-9} requirement would require that the software execute failure-free for several billion hours. Clearly, even if the software had achieved such reliability, one could never assure that the requirement was met. The most reasonable verifiable requirement is in the 10^{-3} to 10^{-4} range.

Many ultra-reliable applications are implemented on relatively small, slow, inexpensive computers. Ultra-reliable applications, such as critical programs could be small (less than 1000 source lines of code) and execute infrequently during an actual mission. With this knowledge, it may be feasible to test the critical program segment on several faster machines, considerably reducing the required test time.

Furthermore, where very high reliability requirements are stated (failure probabilities $<10^{-6}$), they frequently are applicable to a software-controlled process together with its protective and mitigating facilities, and therefore they tend to be overstated if applicable to the process alone. An example of a protective facility is an automatic cutoff system for the primary process and reversion to analog or manual control. An example of a mitigation facility is an automatic sprinkler system that significantly reduces the probability of fire damage in case the software-controlled process generates excessive heat. If the basic requirement is that the probability of extensive fire damage should not exceed 10^{-6} probability of failure per day, and if both protecting and mitigating facilities are in place, it is quite likely that further analysis will show the maximum allowable failure rate for the software controlled process to be on the order of 10^{-3} probability of failure per day, and hence, within the range of current reliability estimation methods. Where the requirements for the software-controlled process still exceed the capabilities of the estimation methodology after allowing for protective and mitigating facilities, fault tolerance techniques may be applied. These may involve fault tolerance (see Hecht and Hecht [B19]) or functional diversity. An example of the latter is to control both temperature and pressure of steam generation, such that neither one of them can exceed safety criteria. The reduction in failure probability that can be achieved by software fault tolerance depends in a large measure on the independence of failure mechanisms for the diverse implementations. It is generally easier to demonstrate the independence of two diverse functions than it is to demonstrate the independence of two computer programs, and hence functional diversity is frequently preferred.

There is an exception to the premise that failure rate gets better with time and upgrades, as shown with the curve in Figure 1. This curve implies that developed and deployed software keeps getting better with

upgrades without consideration for the state of the design and development environment from which the software originated. If the development platforms, software processes, and design environment remain at a constant high level of maturity, or the environment continuously improves in generating quality software, then the actual failure rate performance is expected to follow the curve in Figure 1. If this is not the case and the development environment deteriorates, and the development team loses capability and expertise over time, then the deployed software eventually reaches a point of diminishing return in which the failure rate increases over time where it is no longer possible to correct errors, or correcting errors leads to other errors. At this point, the software is no longer economical to maintain and must be redesigned or recoded.

4. Software reliability assessment and prediction procedure

4.1 General

This clause provides a recommended practice for the practitioner on how to do SR assessment and prediction and what types of analysis can be performed using the technique. It defines a step-by-step procedure for executing SR estimation and describes possible analysis using the results of the estimation procedure.

4.2 Software reliability procedure

The following 13-step procedure for assessing and predicting SR should be executed. Each step of the procedure should be tailored to the project and the current life-cycle phase.

- a) Identify application
- b) Specify the requirement
- c) Allocate the requirement
- d) Make a reliability risk assessment
- e) Define errors, faults, and failures
- f) Characterize the operational environment
- g) Select tests
- h) Select models
- i) Collect data
- j) Estimate parameters
- k) Validate the model
- l) Perform assessment and prediction analysis
- m) Forecast additional test duration

4.2.1 Identify the application

All components of the software application (software product) that are subject to SR assessment or prediction should be identified.

NOTE 1—Example: A diesel generator contains a single computer with a single process that controls the generator. There is only the one component. The goal is to assess the reliability during operation.

NOTE 2—Example: The U.S. Space Shuttle contains many systems. A typical application may contain the following software systems: solid rocket boosters, external tank, orbit processing, navigation, control, launch processing, and reentry processing. Furthermore, each system consists of seven software components. The goal is to specify the reliability of each system and to allocate the reliability to each component.

NOTE 3—This step may be implemented within the Software Requirements Analysis Process in 7.1.2 of IEEE Std 12207™-2008 [B29].

4.2.2 Specify the reliability requirement

The reliability specification should be stated. Components may be combined (for reliability) in series, in parallel, or in combination. For the most conservative analysis and assessment of reliability, a series connection should be used. The reason is that if any component fails, the system fails. Note that this approach is used for prediction. Recognize that the actual connection of software components may not be in succession.

NOTE—This step may be implemented within the Software Requirements Analysis Process in 7.1.2 of IEEE Std 12207-2008 [B29].

4.2.3 Allocate the reliability requirement

Reliability requirements for each component should be allocated. A conservative allocation is a series connection of components. If such is chosen, the Equation (1), Equation (1.1), Equation (1.2), and Equation (1.3) should be carried out, using the following definitions:

- i system identification
- R_T total reliability of system
- R_i system i reliability (assumed equal for each system)
- n number of components in a system
- c_i criticality of system i
- t_i specified execution time of system i
- T_i mean time that system i would survive
- ΔT_i mean time that system i fails to meet execution time specification

The *total* reliability of system i with n components in series is given by Equation (1) as follows:

$$R_T = \prod_{i=1}^n R_i \quad (1)$$

Using Equation (1) and assuming that the system reliabilities R_i are equal, the reliability of each system is given by Equation (1.1) as follows:

$$R_i = R_T^{(1/n)} \quad (1.1)$$

After making the prediction provided by Equation (1.1), estimate the mean time that system i could survive from Equation (1.2) as follows:

$$T_i = R_i t_i = R_T^{(1/n)} t_i \quad (1.2)$$

Furthermore, account for the extent to which the prediction given by Equation (1.2) fails to meet the requirement specification for mean survival time of components. This calculation is provided by Equation (1.3) as follows:

$$\Delta T_i = t_i - T_i \quad (1.3)$$

Consider a hypothetical system consisting of $n = 7$ software components, with criticality c_i and execution time t_i listed in Table 1. The systems have the *total* reliability requirements, R_T , respectively, listed in Table 1. In this example, the *total* reliability R_T is given. In other examples, the objective could be to calculate R_T from Equation (1).

Table 1 also shows two results of the reliability allocation exercise. Namely, the assessed reliability of each system R_i and the *mean time that component i fails to meet execution time requirement* ΔT_i .

Table 1—Hypothetical system example

i	R_T	c_i	t_i (h)	R_i	ΔT_i (h)
1	0.90	5	12	0.9851	0.1793
2	0.91	4	15	0.9866	0.2007
3	0.92	3	20	0.9882	0.2368
4	0.93	2	30	0.9897	0.3094
5	0.94	1	60	0.9912	0.5280
6	0.95	6	10	0.9927	0.0730
7	0.96	7	9	0.9942	0.0523

It is important to see whether there is a “correct relationship” between ΔT_i and the criticality c_i of system i . For example, looking at Figure 2, ΔT_i decreases as c_i increases. This is the correct relationship. If the plot had been otherwise, the software developer should consider examining the SRE process to determine why the correct relationship was not obtained.

NOTE—This step may be implemented within the Software Architectural Design Process in 7.1.3 of IEEE Std 12207-2008 [B29].

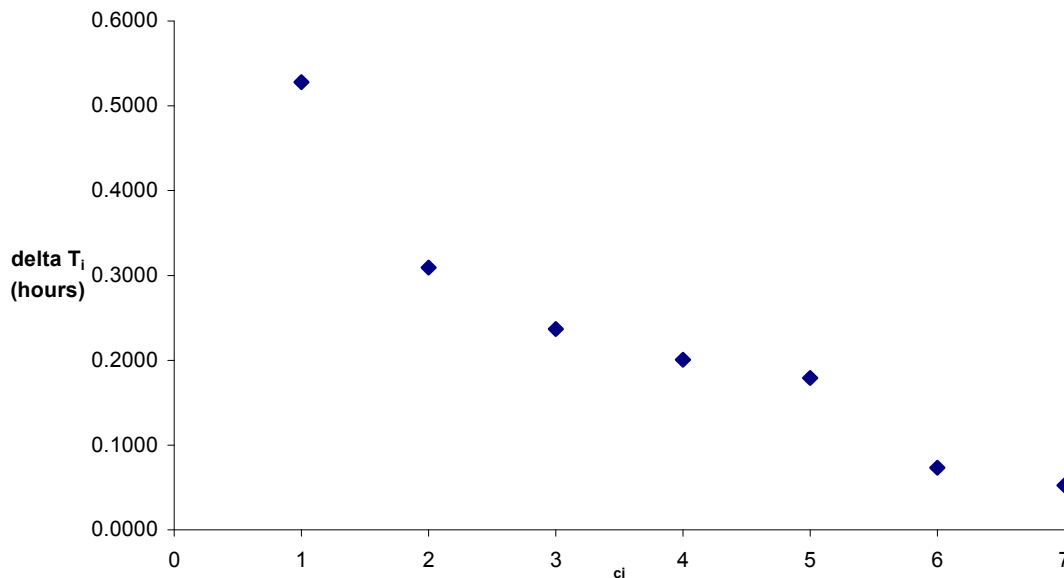


Figure 2—Time that system fails to meet specified execution time requirement, delta versus system criticality c_i

4.2.4 Make a reliability risk assessment

A reliability risk assessment should be based on the risk to reliability due to software defects or errors caused by requirements and requirements changes. The method to ascertain risk based on the number of requirements and the impact of changes to requirements is inexact, but nevertheless, necessary for early design assessments of large scale systems. It is probably not necessary for smaller systems and modules. Risk assessment is performed by calculating two quantities of the risk, the risk for remaining failures, and risk for the time to next failure. The methods to calculate these two risk metrics for risk assessment are provided in 5.3.1.12.2.

NOTE—This step may be implemented within the Risk Management Process in 6.3.4 of IEEE Std 12207-2008 [B29].

4.2.5 Define errors, faults, and failures

Project specific definitions for error, failures, and faults should be provided within the constraints of the definitions in 2.4, 2.6, and 2.9, respectively. These definitions are usually negotiated by the testers, developers, and users. These definitions should be agreed upon prior to the beginning of test. There are often commonalities in the definitions among similar products (i.e., most people agree that a software fault that stops all processing is a failure). The important consideration is that the definitions be consistent over the life of the project. There are a number of considerations relating to the interpretation of these definitions. The analyst must determine the answers to the following questions as they relate to errors, faults, and failures:

- Is an error, failure, or fault counted if it is decided not to seek out and remove the cause of the error, failure, or fault?
- Are repeated errors, failures, or faults counted? If so, how are they counted (e.g., one pattern failure represents several failure occurrences of the same failure type)?
- What is an error, failure, or fault in a fault-tolerant system?
- Is a series of errors, failures, or faults counted if it is triggered by data degradation?

NOTE 1—A discussion of each of these considerations is provided in pp. 77–85 in *Software Reliability: Measurement, Prediction, Application* [B54].

NOTE 2—Projects need to classify failures by their severity. An example classification is provided in the NOTE of 2.8. The classes are usually distinguished by the criticality of the failure to system safety and/or cost of fault correction. It is desirable to consider failure severity by type.

NOTE 3—For some projects, there appears to be relative consistency of failures. For example, if 10% of the failures occurring early in test fall in a particular severity class, about the same percentage will be expected to be found in that class late in test.

NOTE 4—Time measurement based on CPU time or failure counts based on CPU time for failure data are preferred. However, there are approximating techniques, if the direct measurement of CPU time is not available (see pp. 156–158 in *Software Reliability: Measurement, Prediction, Application* [B54]). Failure times should be recorded in execution time or failure counts per execution time interval. However, should execution time not be available, elapsed clock time is a satisfactory approximation. When failure times are collected from multiple machines operating simultaneously, intervals between failures should be counted by considering all machines operating simultaneously. If the machines have different average instruction execution rates, execution times (reciprocal of execution rate) should be adjusted by using the average of the average execution rate.

NOTE 5—This step may be implemented within the Measurement Process in 6.3.7 of IEEE Std 12207-2008 [B29].

4.2.6 Characterize the operational environment

The operational environment should be characterized. This characterization should be based on the following three aspects:

- System configuration—the arrangement of the system's components. Software-based systems include hardware as well as software components.
- System evolution.
- System operational profile.

NOTE 1—The purpose of determining the system configuration is twofold:

- To determine how to allocate system reliability to component reliabilities
- To determine how to combine component reliabilities to establish system reliability

NOTE 2—In modeling SR, it is necessary to recognize that systems frequently evolve as they are tested. That is, new code or even new components are added. Special techniques for dealing with evolution are provided in pp. 166–176 in *Software Reliability: Measurement, Prediction, Application* [B54].

NOTE 3—A system may have multiple operational profiles or operating modes. For example, a space vehicle may have ascent, on-orbit, and descent operating modes. Operating modes may be related to the time of operation, installation location, and customer or market segment. Reliability should be assessed separately for different modes if they are significantly different.

NOTE 4—This step may be implemented within the Stakeholder Requirements Definition Process in 6.4.1 of IEEE Std 12207-2008 [B29].

4.2.7 Select tests

Tests and test cases should be selected to reflect how the system will actually be used. Test selection criteria and guidance on the appropriateness of one test over another depending on certain situations (test conditions or use conditions) may be useful.

The tester should select one of the following approaches:

- The test duplicates the actual operational environments (as closely as possible).
- Testing is conducted under severe conditions for extended periods of time.
- The reliability modeling effort must take into account the approach taken by the test team to expose faults so that accurate assessments can be made.

NOTE—This step may be implemented within the System Qualification Testing Process in 6.4.6 of IEEE Std 12207-2008 [B29].

4.2.8 Select models

One or more SR models should be selected and applied. Model selection criteria and guidance on the appropriateness of one model over another depending on certain situations may be useful.

NOTE—This step may be implemented within the Decision Management and Measurement Processes in 6.3.3 and 6.3.7, respectively, of IEEE Std 12207-2008 [B29].

The models described in Clause 5 provided proven accurate results in specific environments, but there is no guarantee that these models will be suitable in all environments; therefore, it is necessary that users compare several models applicable to the user's environment prior to final selection.

4.2.9 Collect data

Data collection should be sufficient to determine if the objectives of the SR effort have been met.

In setting up a reliability program, the following goals should be achieved:

- Clearly defined data collection objectives
- Collection of appropriate data (i.e., data geared to making reliability assessments and predictions)
- Review of the collected data promptly to see ensure that it meets the objectives

A process should be established addressing each of the following steps, which are further described in 5.6.1:

- a) Establish the objectives
- b) Set up a plan for the data collection process
- c) Apply tools
- d) Provide training
- e) Perform trial run
- f) Implement the plan
- g) Monitor data collection
- h) Evaluate the data as the process continues
- i) Provide feedback to all parties

NOTE—This step may be implemented within the Measurement Process in 6.3.7 of IEEE Std 12207-2008 [B29].

4.2.10 Estimate parameters

A parameter estimation method should be selected. Three common methods of parameter estimation exist: method of moments, least squares, and maximum likelihood. Each of these methods has useful attributes. Maximum likelihood estimation is the recommended approach.

NOTE 1—This step may be implemented within the Measurement Process in 6.3.7 of IEEE Std 12207-2008 [B29].

NOTE 2—A full treatment of parameter estimation is provided by Farr [B11] and in *Software Reliability: Measurement, Prediction, Application* [B54] and *Software Engineering: Design, Reliability, and Management* [B69]. All of the SRE tools described in Annex B perform parameter estimation, using one or more of these methods.

4.2.11 Validate the model

The chosen model or models should be validated. The model validation should also include checking the data collection and parameter estimation methods.

NOTE 1—Several considerations are involved in properly validating a model for use on a given production project. First, it is necessary to deal with the assumptions of the model under evaluation. Choosing appropriate failure data items and relating specific failures to particular intervals of the life-cycle or change increments often facilitate this task (see Schneidewind and Keller [B59]). Depending on the progress of the *production* project, the model validation data source should be selected from the following, listed in the order of preference:

- a) Production project failure history (if project has progressed sufficiently to produce failures)
- b) Prototype project employing similar products and processes as the production project
- c) Prior project employing similar products and processes as the production project (reference Annex B)

NOTE 2—Using one of these data sources, the analyst should execute the model several times within the failure history period and then compare the model output to the actual subsequent failure history using one of the following:

- a) Predictive validity criteria (5.2.2)
- b) A traditional statistical goodness-of-fit test [e.g., chi-square or Kolmogorov–Smirnov (K-S)]

NOTE 3—It is important that a model be continuously rechecked for validation, even after selection and application, to ensure that the fit to the observed failure history is still satisfactory. In the event that a degraded model fit is experienced, alternate candidate models should be evaluated using the procedure of 4.2.7 through 4.2.10.

NOTE 4—This step may be implemented within the Measurement Process in 6.3.7 of IEEE Std 12207-2008 [B29].

4.2.12 Perform assessment and prediction analysis

Once the data has been collected and the model parameters estimated, the analyst is ready to perform the appropriate analysis. This analysis should be to assess the current reliability of the software, predict the number of failures remaining in the code, or predict a test completion date.

4.2.12.1 Recommended analysis practice

Subclauses 4.2.12.2 and 4.2.12.3 provide details of analysis procedures that are supported by SRE technology.

4.2.12.2 Assess current reliability

Reliability assessments in test and operational phases basically follow the same procedure. However, there is a difference. During the test phase, software faults should be removed as soon as the corresponding software failures are detected. As a result, reliability growth (e.g., decreasing failure rate over time) should be observed. However, in the operational phase, correcting a software fault may involve changes in multiple software copies in the customers' sites, which, unless the failure is catastrophic, is not always done until the next software release. Therefore, the software failure rate may not be decreasing.

4.2.12.3 Predict achievement of a reliability goal

The date at which a given reliability goal can be achieved is obtainable from the SR modeling process illustrated in Figure 3. As achievement of the reliability target approaches, the adherence of the actual data to the model predictions should be reviewed and the model corrected, if necessary. Refer to Annex C.

NOTE—This step may be implemented within the Decision Management and Measurement Processes in 6.3.3 and 6.3.7, respectively, of IEEE Std 12207-2008 [B29].