IEEE STANDARDS ASSOCIATION

IEEE Recommended Practice on Software Reliability

IEEE Reliability Society

Sponsored by the
Standards Committee



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

IEEE Std 1633™-2016
(Revision of IEEE Std 1633-2008)

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

IEEE Std 1633™-2016
(Revision of IEEE Std 1633-2008)

IEEE Recommended Practice on Software Reliability

Sponsor

Standards Committee
of the
IEEE Reliability Society

Approved 22 September 2016

IEEE-SA Standards Board

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

This is a Redline Document produced by Techstreet, a business of Thomson Reuters.

This document is intended to provide users with an indication of changes from one edition to the next. It includes a full-text version of the new document, plus an indication of changes from the previous version.

Redlines are designed to save time and improve efficiencies by using the latest software technology to find and highlight document changes. More professionals are using valuable new technologies like redlines, to help improve outcomes in a fastpaced global business world.

Because it may not be technically possible to capture all changes accurately, it is recommended that users consult previous editions as appropriate. In all cases, only the current base version of this publication is to be considered the official document.

Documents Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

Redline Processing Notes:

1. ~~**Red Text**~~ - Red strikethrough text denotes deletions.

2. **Blue Text** - Blue underlined text denotes modifications and additions.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

Abstract: The methods for assessing and predicting the reliability of software, based on a life-cycle approach to software reliability engineering (SRE), 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: IEEE 1633™, software failure modes, software reliability

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

Copyright © 2017 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 18 January 2017. 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 Integrated and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

COCOMO is a registered trademark of Barry W. Boehm.

Excel is a registered trademark of Microsoft Corporation in the United States and/or other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States and other countries.

Price is a registered trademark of Price Systems, L.L.C.

217Plus is a trademark of Quanterion Solutions Incorporated.

PDF: ISBN 978-1-5044-3648-9 STD22370
Print: ISBN 978-1-5044-3649-6 STDPD22370

IEEE prohibits discrimination, harassment, and bullying.
For more information, visit http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html.
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.

Copyright © 2017 IEEE. All rights reserved.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading "Important Notices and Disclaimers Concerning IEEE Standards Documents." They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/IPR/disclaimers.html.

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association ("IEEE-SA") Standards Board. IEEE ("the Institute") develops its standards through a consensus development process, approved by the American National Standards Institute ("ANSI"), which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed through scientific, academic, and industry-based technical working groups. Volunteers in IEEE working groups are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, 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.

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied "AS IS" and "WITH ALL FAULTS."

Use of an IEEE standard is wholly voluntary. The existence of an IEEE standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own 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.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Copyright © 2017 IEEE. All rights reserved.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of 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 of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive 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 comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

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

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document 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

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted 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 these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Copyright © 2017 IEEE. All rights reserved.
Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, 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.

Updating of IEEE Standards documents

Users of IEEE Standards documents 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.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, 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 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 Xplore at http://ieeexplore.ieee.org/ or contact IEEE at the address listed previously. For more information about the IEEE-SA or IEEE's standards development process, visit the IEEE-SA Website at http://standards.ieee.org.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE-SA Website at the following URL: http://standards.ieee.org/findstds/errata/index.html. Users are encouraged to check this URL for errata periodically.

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 by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at http://standards.ieee.org/about/sasb/patcom/patents.html. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. 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 standard 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.

Copyright © 2017 IEEE. All rights reserved.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

Participants

At the time this IEEE recommended practice was completed, the IEEE 1633 Working Group had the following membership:

Ann Marie Neufelder, *Chair*
Martha Wetherholt, *Vice Chair*
Debra Haehn, *Secretary*
Lou Gullo, *Sponsor Chair*

Jacob Axman
Bakul Banerjee
David Bernreuther
Nematollah Bidokhti
Robert Binder
Sonya Davis
Mary Ann DeCicco
Lance Fiondella
Willie Fitzpatrick
Kevin Frye
Loren Garroway
Richard E. Gibbs III
Michael Grottke
Darwin Heiser
Nathan Herbert
Claire Jones
Burdette Joyner
Ahlia T. Kitwana
Peter Lakey
Ming Li
Andy Long
Debra Greenhalgh Lubas
Andrew Mack
Franklin Marotta
Kevin Mattos
Brian McQuillan
Rajesh Murthy
Allen Nikora
Mark Ofori-kyei
Robert Raygan
Ying Shi
Marty Shooman
Mark Sims
Michael Siok
Shane Smith
George Stark
Kishor Trivedi
Thierry Wandji
Martin Wayne
Yuan Wei
Harry White

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

Johann Amsenga
Bakul Banerjee
Pieter Botman
Bill Brown
Keith Chow
Paul Croll
Sonya Davis
Mary Ann DeCicco
Neal Dowling
Richard Doyle
Lance Fiondella
Debra Greenhalgh
Randall Groves
Louis Gullo
Debra Haehn
Jon Hagar
Werner Hoelzl
Bernard Homes
Noriyuki Ikeuchi
Cheryl Jones
Piotr Karocki
Chad Kiger
Ahlia Kitwana
Edward McCall
Jeffrey Moore
Rajesh Murthy
Andrew Nack
Ann Marie Neufelder
Michael Newman
Mark Ofori-Kyei
Howard Penrose
Iulian Profir
Stephen Schwarm
Jeremy Smith
Thomas Starai
Eugene Stoudenmire
Walter Struppler
Eric Thibodeau
Martha Wetherholt
Paul Work
Daidi Zhong

Copyright © 2017 IEEE. All rights reserved.
Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

When the IEEE-SA Standards Board approved this recommended practice on 22 September 2016, it had the following membership:

Jean-Philippe Faure, *Chair*
Ted Burse, *Vice Chair*
John D. Kulick, *Past Chair*
Konstantinos Karachalios, *Secretary*

Chuck Adams
Masayuki Ariyoshi
Stephen Dukes
Jianbin Fan
Ronald W. Hotchkiss
J. Travis Griffith
Gary Hoffman
Michael Janezic
Joseph L. Koepfinger*
Hung Ling
Kevin Lu
Gary Robinson
Annette D. Reilly
Mehmet Ulema
Yingli Wen
Howard Wolfman
Don Wright
Yu Yuan
Daidi Zhong

*Member Emeritus

Copyright © 2017 IEEE. All rights reserved.
Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Introduction

This introduction is not part of IEEE Std 1633-2016, IEEE Recommended Practice on Software Reliability.

Software is, from a materials viewpoint, both malleable and ductile. This means there are multiple ways to introduce failures, intentional and un-intentional. Fixing a software defect can introduce a potential defect. In many cases the failures that result from software defects are both predictable and avoidable but they still occur because of the following:

a) Lack of available calendar time/resources to find all of the defects that can result in failures

b) Exceedingly complex event driven systems that are difficult to conceptualize and therefore implement and test

c) Organizational culture that neglects to support sufficient rigor, skills, or methods required to find the defects

d) Technical decisions that result in incorrect architecture or design decision that cannot support the stakeholders specifications

e) Insufficient project or risk management that leads to schedule delays that lead to less time for reliability testing

f) Operations—Contract issues, interoperability due to bad specifications and stakeholder communications

Even a small number of software failures can lead to monetary catastrophes such as a cancelled project. Hardware (HW) failures can be random, due to wear-out or the result of a systematic design flaw. Reliability maintainability availability (RMA) is used to prevent and deal with hardware failures. Software failures may result from systematic flaws in the requirements, design, code or interfaces. Hence, software failure does not require an RMA but instead a corrective action to an existing installation. Software failures can be common cause failures in that the same failure mode can cause multiple failures in more than one part of the software.

Software reliability engineering (SRE) is an established discipline that can help organizations improve the reliability of their products and processes. It is important for an organization to have a process discipline if it is to produce high reliability software. These are specific practices and recommendations, each of which has a context within the software engineering life cycle. A specific practice may be implemented or used in a particular stage of the life cycle or used across several stages. Figure 1 shows how the focus of SRE shifts as a project progresses from inception to release. The size of each bubble on this figure corresponds to how much the particular SRE practices are being executed during each particular phase of development or operation. For example in software engineering projects, the failure modes and effects analysis (FMEA) is typically performed earlier in the life cycle.

Copyright © 2017 IEEE. All rights reserved.
Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

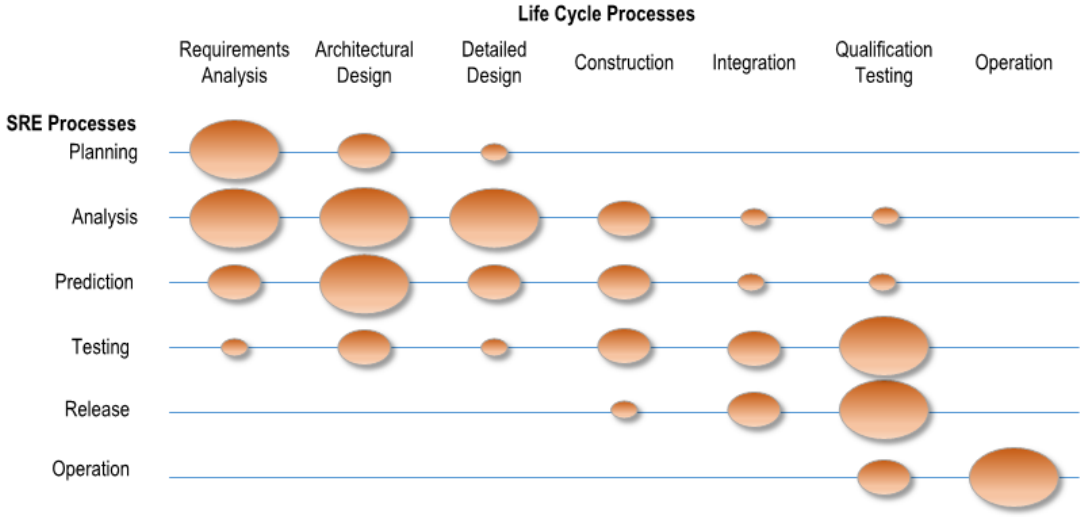


Figure 1—SRE focus by stage

The scope of this recommended practice is to address software reliability (SR). It does not specifically address systems reliability, software safety, or software security. However, it does recognize that safety and security requirements are part of the initial risk assessment. 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 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 and analyzing requirements, and continuing into the implementation.

This standard includes guidance on the following:

- Common terminology
- Assessment of software reliability risks that pertain to the software or project
- Software failure mode analyses that can help to identify and reduce the types of defects most likely to result in a system failure
- Models for predicting software reliability early in development
- Models for estimating software reliability in testing and operation
- Test coverage and test 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
- Identifying elements in a software system that are leading candidates for redesign to improve reliability

Copyright © 2017 IEEE. All rights reserved.
Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Revisions to the document and notes

- This document is a revision of IEEE Std 1633-2008.
- Addition of models that can be used early in development, before testing, to predict software reliability
- Addition of failure modes analysis
- Revision of the software reliability growth models so as to be more practical
- Addition of practical guidance for selecting the best models
- Addition of techniques for applying SRE in incremental development
- Addition of methods to assess the SRE related risks

Structure of the recommended practice

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

- Clause 1 provides the overview.
- Clause 2 provides the normative references.
- Clause 3 provides the definitions, acronyms, and abbreviations
- Clause 4 provides the roles, approaches, and concepts related to SRE.
- Clause 5 provides the SRE procedures.
- Clause 6 provides the predictive and estimation SRE models.
- Annex A contains software failure modes effects analysis (SFMEA) templates.
- Annex B provides methods for predicting EKSLOC (Effective 1000 Source Lines of Code), which is necessary for the reliability predictions as well as for models that predict software defect density and defects prior to the testing phase.
- Annex C provides additional software reliability growth models and provides the results of a survey of most used software reliability growth models.
- Annex D provides the estimated cost of the SRE tasks.
- Annex E contains a list of tools that pertain to SRE tasks.
- Annex F contains examples.
- Annex G contains an informative Bibliography.

Copyrights and Permissions

Permissions have been granted as follows:[1]

Content appearing in Clause 4 Roles, approach, concepts, including all subclauses; 5.4.3 Measure test coverage, 5.5 Support release decision; adapted with permission of Robert V. Binder, *Beware of Greeks bearing data*, 2014.

[1] Every effort has been made to secure permission to reprint borrowed material contained in this document. If omissions have been made, please bring them to our attention.

Copyright © 2017 IEEE. All rights reserved.
Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Content and tables appearing in 5.1.1.1, 5.1.1.3, 5.1.3.4, 5.1.3.5, 5.3.1, 5.3.2, 5.3.3, 5.3.5.2, 5.3.5.3, 5.3.8, 5.4.7, 6.1, 6.2.1.1, 6.2.1.2, 6.2.2.1, B.2.1, B.2.3, B.3, Table 12 "Keywords associated with common root causes for defects," Annex D, F.3, F.4.4, F.4.5, Table 48 "Average defect densities by application type (EKSLOC)," Table 45 "Factors in determining root cause inaccuracies" reprinted with permission of Ann Marie Neufelder, Softrel, LLC "Software Reliability Toolkit" © 2015.

Content and tables appearing in 5.4.5, 6.3.2, 6.3.3, F.3, F.6 reprinted with permission of Ann Marie Neufelder, Softrel, LLC "Advanced Software Reliability" © 2015.

Figure 27 "SFMEA process," 5.2.2, F.4.3, and all tables in Annex A reprinted with permission from Ann Marie Neufelder, Softrel, LLC "Effective Application of Software Failure Modes Effects Analysis" © 2014.

Table 8 "Relationship between risks and outcome" reprinted with permission of Softrel, LLC. "Four things that are almost guaranteed to reduce the reliability of a software intensive system," Huntsville Society of Reliability Engineers RAMS VII Conference © 2014.

A portion of 5.5.1 has been reprinted with permission from Lockheed Martin Corporation article entitled "Determine Release Stability" © 2015 Lockheed Martin Corporation. All rights reserved.

A portion of 5.5.4 has been reprinted with permission from Lockheed Martin Corporation article entitled "Perform a Reliability Demonstration Test (RDT)" © 2015 Lockheed Martin Corporation. All rights reserved.

Table 47 Shortcut Model Survey and Table F.9 Example of the Shortcut Model Survey reprinted with permission Softrel, LLC "A Practical Toolkit for Predicting Software Reliability" © 2006.

Table 49 reprinted with permission from Capers Jones, "Software Industry Blindfolds: Invalid Metrics and Inaccurate Metrics," Namcook Analytics, 2005.

"Elevator Example" in F.4.2 reprinted with permission from Peter B. Lakey, *Operational Profile Testing* © 2016.

Copyright © 2017 IEEE. All rights reserved.
Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Contents

1. Overview 14
   1.1 Scope 14
   1.2 Purpose 14

2. Normative references 14

3. Definitions, acronyms, and abbreviations 15
   3.1 Definitions 15
   3.2 Acronyms and abbreviations 18

4. Role, approach, and concepts 21
   4.1 What is software reliability engineering? 21
   4.2 Strategy 21
   4.3 Project-specific SRE tailoring 23
   4.4 Life-cycle considerations for SRE 30

5. Software reliability procedure 34
   5.1 Plan for software reliability 34
   5.2 Develop failure modes model 56
   5.3 Apply software reliability during development 67
   5.4 Apply software reliability during testing 110
   5.5 Support release decision 136
   5.6 Apply software reliability in operation 142

6. Software reliability models 148
   6.1 Overview 148
   6.2 Models that can be used before testing 149
   6.3 Models that can be used during and after testing 159

Annex A (informative) Software failure modes effects analysis templates 168
   A.1 Templates for preparing the software failure modes effects analysis (SFMEA) 168
   A.2 Templates for analyzing the failure modes and root causes 170
   A.3 Template for consequences 174
   A.4 Template for mitigation 175

Annex B (informative) Methods for predicting software reliability during development 176
   B.1 Methods for predicting code size 176
   B.2 Additional models for predicting defect density or defects 180
   B.3 Factors that have been correlated to fielded defects 186

Annex C (informative) Additional information on software reliability models used during testing 189
   C.1 Models that can be used when the fault rate is peaking 189
   C.2 Models that can be used when the fault rate is decreasing 189
   C.3 Models that can be used with increasing and then decreasing fault rate 194
   C.4 Models that can be used regardless of the fault rate trend 196
   C.5 Models that estimate remaining defects 197
   C.6 Results of the IEEE survey 198

Annex D (informative) Estimated relative cost of SRE tasks 200

Annex E (informative) Software reliability engineering related tools 203

Copyright © 2017 IEEE. All rights reserved.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

Annex F (informative) Examples 205
F.1 Examples from 5.1 205
F.2 Examples from 5.2 209
F.3 Examples from 5.3 220
F.4 Examples from 5.4 230
F.5 Examples from 5.5 251
F.6 Examples from 5.6 252

Annex G (informative) Bibliography 254

Copyright © 2017 IEEE. All rights reserved.

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

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

This recommended practice defines the software reliability engineering (SRE) processes, prediction models, growth models, tools, and practices of an organization. This document and its models and tools are useful to any development organization to identify the methods, equations, and criteria for quantitatively assessing the reliability of a software or firmware subsystem or product. Organizations that acquire software subsystems or products developed with consideration to this recommended practice will benefit by knowing the reliability of the software prior to acquisition. This document does not seek to certify either the software or firmware or the processes employed for developing the software or firmware.

~~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).~~

Documents
Document Sharing Hub

Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.

Full document details: IEEE 1633-2016

1.2 Purpose

The purpose for assessing the reliability of a software or firmware subsystem or product is to determine whether the software has met an established reliability objective and facilitate improvement of product reliability. The document defines the recommended practices for predicting software reliability (SR) early in development so as to facilitate planning, sensitivity analysis and trade-offs. This document also defines the recommended practices for estimating SR during test and operation so as to establish whether the software or firmware meets an established objective for reliability.

~~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~~

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

- 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.

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

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

- 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.

- 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. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

IEEE Std 12207™-2008, ISO/IEC/IEEE Standard for Systems and Software Engineering—Software Life Cycle Processes. [1, 2]

3. Definitions, acronyms, and abbreviations

For the purposes of this ~~recommended practice~~ document, the following terms and definitions apply. The ~~*Authoritative Dictionary of* *IEEE Standards Terms* [B22] and IEEE Std 610.12™-1990 [B23]~~ *Dictionary Online* should be ~~referenced~~ consulted for terms not defined in this clause. [3]

3.1 Definitions

agile (software development): (A) IEEE definition: software development approach based on iterative development, frequent inspection and adaptation, and incremental deliveries, in which requirements and solutions evolve through collaboration in cross-functional teams and through continuous stakeholder feedback. **(B)** Agile Consortium definition: software development approach based on iterative development, frequent inspection and adaptation, and incremental deliveries, in which requirements and solutions evolve through collaboration in cross-functional teams and through continuous stakeholder feedback.

assessment: **(A)** An action for applying specific documented criteria to a specific software module, package, or product for the purpose of determining acceptance or release of the software module, package, or product. (ISO/IEC/IEEE 24765™-2010) **(B)** Determining what action to take for software that fails to meet goals (e.g., intensify inspection, intensify testing, redesign software, and revise process).

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

[2] IEEE publications are available from The Institute of Electrical and Electronics Engineers (http://standards.ieee.org/).

[3] *IEEE Standards Dictionary Online* is available at: http://ieeexplore.ieee.org/.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

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~~ 4

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

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

defect: (A) A problem that, if not corrected, could cause an application to either fail or to produce incorrect results. (ISO/IEC/IEEE 24765-2010)

NOTE—For the purposes of this standard, defects are the result of errors that are manifest in the system requirements, software requirements, interfaces, architecture, detailed design, or code. A defect may result in one or more failures. It is also possible that a defect may never result in a fault if the operational profile is such that the code containing the defect is never executed.

defect pileup: When residual defects in the software are not removed, and over time, the number increases to the point of adversely affecting the reliability and schedule of the software releases.

error: ~~**(A)** A discrepancy between a computed, observed, or measured value or condition and the true, specified or theoretically correct value or condition. **(B)**~~ A human action that produced an incorrect result, such as ~~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. (ISO/IEC/IEEE 24765-2010)

NOTE—For the purposes of this standard, an *error* can also include incorrect software interfaces, software architecture, design, or code.

4 ~~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~~ standard.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

evolutionary development: Developing and delivering software in iterative drops to the customer with a concentration on developing the most important or least understood requirements first, and once those are working and approved by the customer, improve the requirements, design, and testing to increase the functionality to meet the stakeholders' desired functionality and performance. This is especially useful for development of new products. *See also:* **incremental development**.

NOTE—See Larman [B51].[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.

NOTE—Processor time is usually less than elapsed time because the processor may be idle (for example, awaiting needed computer resources) or employed on other tasks during the execution of a program.

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 product to perform a required function or its inability ~~functional unit~~ to perform within previously specified limits ~~its required function~~. (ISO/IEC/IEEE 24765-2010) **(C)** A departure of program operation from program requirements. A *failure* may be produced when a fault is encountered and a loss of the expected service results.

NOTE 1—A *failure* may be produced when a fault is encountered and a loss of the expected service to the user results.

NOTE 2—There may not be a one-to-one relationship between faults and failures. This can happen if the system has been designed to be fault tolerant. It can also happen if a fault does not result in a failure either because it is not severe enough to result in a failure or does not manifest into a failure due to the system not achieving that operational or environmental state that would trigger it.

failure intensity: Total failures observed over total operational hours experienced.

[5] The numbers in brackets correspond to those of the bibliography in Annex G.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

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~~ *Syn:* **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.

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.~~

fault: (A) A defect in the code that can be the cause of one or more failures. **(B)** A manifestation of an error in the software. (ISO/IEC/IEEE 24765-2010) ~~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.~~

NOTE—There may not necessarily be a one-to-one relationship between faults and failures if the system has been designed to be fault tolerant or if a fault is not severe enough to result in a failure.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

fault tolerance: **(A)** The survival attribute of a system that allows it to deliver the required service after faults have manifested themselves within the system. **(B)** The ability of a system or a component to continue normal operation despite the presence of hardware or software faults. (ISO/IEC/IEEE 24765-2010)

firmware: The combination of a hardware device, software, and data that are incorporated into the hardware device.

NOTE—For compliance with this standard, *firmware* is treated as software in a programmable device. (Adapted from IEEE Std 610.12™-1990)

function point: (A) A unit of measurement to express the amount of business functionality an information system (as a product) provides to a user. **(B)** A unit that expresses the size of an application or of a project (ISO/IEC/IEEE 24765-2010) *Function points* measure software size. The functional user requirements of the software are identified, and each one is categorized into one of five types: outputs, inquiries, inputs, internal files, and external interfaces.

incremental development: A software development technique in which requirements definition, design, implementation, and testing occur in an overlapping, iterative (rather than sequential) manner, resulting in incremental completion of the overall software product. *Contrast:* **waterfall model**.

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

life-cycle model: A framework containing the processes, activities, and tasks involved in the development, operation, and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use. (ISO/IEC/IEEE 24765-2010) *Contrast:* **software development cycle**.

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

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

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.

operational: (A) Pertaining to the status given a software product once it has entered the operation and maintenance phase. **(B)** Pertaining to a system or component that is ready for use in its intended environment.

reliability growth: (A) The amount the software reliability improves from operational usage and other stresses. **(B)** The improvement in reliability that results from correction of faults.

~~**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.~~

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

scrum: (A) An iterative and incremental agile software development methodology for managing product development. **(B)** The iterative project management framework used in agile development, in which a team agrees on development items from a requirements backlog and produces them within a short duration of a few weeks.

sprint: (A) The basic unit of development in agile/scrum. The *sprint* is restricted to a specific duration. The duration is fixed in advance for each sprint and is normally between one week and one month, with two weeks being the most common. **(B)** The short time frame, in which a set of software features is developed, leading to a working product that can be demonstrated to stakeholders.

software development cycle: (A) The period of time that begins with the decision to develop a software product and ends when the software is delivered. (ISO/IEC/IEEE 24765-2010) For the development part of the software life-cycle processes this would include practices for planning, creating, testing, and deploying a software system.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

software engineering: (A) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is the application of engineering to software. **(B)** The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software. (ISO/IEC/IEEE 24765-2010)

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~~ the user's 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. **(E)** Capability of the software product to satisfy stated and implied needs when used under specified conditions. (ISO/IEC/IEEE 24765-2010)

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~~ defects in the software. The inputs to the system determine whether existing ~~faults~~ defects, if any, are encountered.

software reliability engineering (SRE): (A) The application of statistical techniques to data collected during system development and operation to specify, estimate, or assess the reliability of software-based systems. **(B)** The application of software reliability best practices to enhance software reliability characteristics of software being developed and integrated into a system.

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.

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.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

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.

NOTE— Reliability predictions are a measure of the probability that the software will perform without failure over a specific interval, under specified conditions.

3.2 Acronyms and abbreviations

API	Application Programmers Interface
ASIC	Application Specific Integrated Circuit
BIOS	Basic Input Output System
BOM	bill of materials
CASRE	computer-aided software reliability estimates
CIL	critical items list
CMMI®	Capability Maturity Model Integration® 6
COTS	commercial-off-the-shelf software
CR	capture/recapture
CSCI	computer software configuration item
CTMC	Continuous Time Markov Chain
DD	defect density
DDL	dynamically linked libraries

6 Capability Maturity Model Integrated and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of these products. Equivalent products may be used if they can be shown to lead to the same results.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

DDN	defect days number
DLOC	defects per line of code
EFSM	extended finite state machine
EKSLOC	Effective 1000 Source Lines of Code
FD	fault days
FDIR	fault/failure detection, isolation, and recovery
FDSC	failure definition and scoring criteria
FI	fault injection
FMEA	failure modes and effects analysis
FOSS	free open source software
FPGA	field programmable gate array
FTA	fault tree analysis
FW	firmware
GFS	government furnished software
GO	Goel-Okumoto
HW	hardware
IDD	interface design document
IR	intermediate representations
JM	Jelinski-Moranda

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

JVM	Java™ Virtual Machine[7]
kB	kilobytes
KSLOC	1000 source lines of code
LCM	life-cycle model
LRU	line replaceable unit
MCDC	multiple condition decision coverage
MCUM	Markov Chain Usage Model
MTBCF	mean time between critical failure
MTBEFF	mean time between essential function failure
MTBF	mean time between failure
MTBSA	mean time between system abort
MTSWR	mean time to software restore
MTTR	mean time to repair
OP	operational profile
OS	operating system
PHA	preliminary hazards analysis
PIE	propagation, infection, execution

[7] Java is a trademark of Sun Microsystems, Inc. in the United States and other countries. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE of these products. Equivalent products may be used if they can be shown to lead to the same results.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

RBD reliability block diagram

RCA root cause analysis

RCM reliability criticality metric

RDT Reliability Demonstration Test

RePs reliability prediction system

RFP request for proposal

RG reliability growth

RMA reliability maintainability availability

RPN risk priority number

RT requirements traceability

SCODE software code

SDD software design document

SDP software development plan

SFMEA software failure modes effects analysis

SFTA software fault tree analysis

SLOC source lines of code

SR software reliability

SRE software reliability engineering

SRGM Software Reliability Growth Model

SRM Software Reliability Model

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

SRPP	software reliability program plan
SRS	software requirements specification
SUT	system under test
SW	software
SWDPMH	software defects per million hours
SyRs	system requirements specification
TNI	trouble not identified
UI	user interface
UPM	Unified Process Model
WG	Working Group

4. Role, approach, and concepts

4.1 What is software reliability engineering?

IEEE Std 610.12-1990 defines reliability as: "The ability of a system or component to perform its required functions under stated conditions for a specified period of time." *Software reliability engineering* (SRE) supports cost-effective development of high-reliability software.

While reliability is a critical element of software quality, it is not the only one. SRE does not directly address modeling or analysis of other important quality attributes, including usability, security, or maintainability. There are many proven practices and technologies that address the full spectrum of software quality as well as specific aspects. SRE should be applied in concert with them.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

SRE can include analyses to prevent or remove software defects that lead to failures. SRE can also provide unambiguous and actionable information about likely operational reliability throughout the software development cycle. Just as the accuracy of weather forecasts is generally better for the very near term, SRE predictions are more accurate as the software development nears completion.

4.2 Strategy

SRE allows developers to have confidence that a software system will meet or exceed its quantitative and qualitative reliability requirements. SRE uses a data-driven strategy to achieve this, which includes but is not limited to the following process areas, analyses, and metrics:

a) **Planning**—Determine the scope of the software reliability (SR) program based on the availability of resources and the needs of the project. Identify the software line replaceable units (LRUs) that will be included in the SRE.

b) **Analyzing**—Assess the software related risks and perform the failure mode analyses early to develop requirements and design that support more reliable software.

c) **Prediction**—Predict SR in accordance with the system operational profile (OP) and performing the sensitivity analysis early to identify appropriate goals for the SR and identify weaknesses in the planned development activities that could affect the reliability of the software. Define requirements for acceptable failure rate, residual defects, defect backlog, availability, or other quantitative measure taking into account the criticality and interface and architectural dependencies.

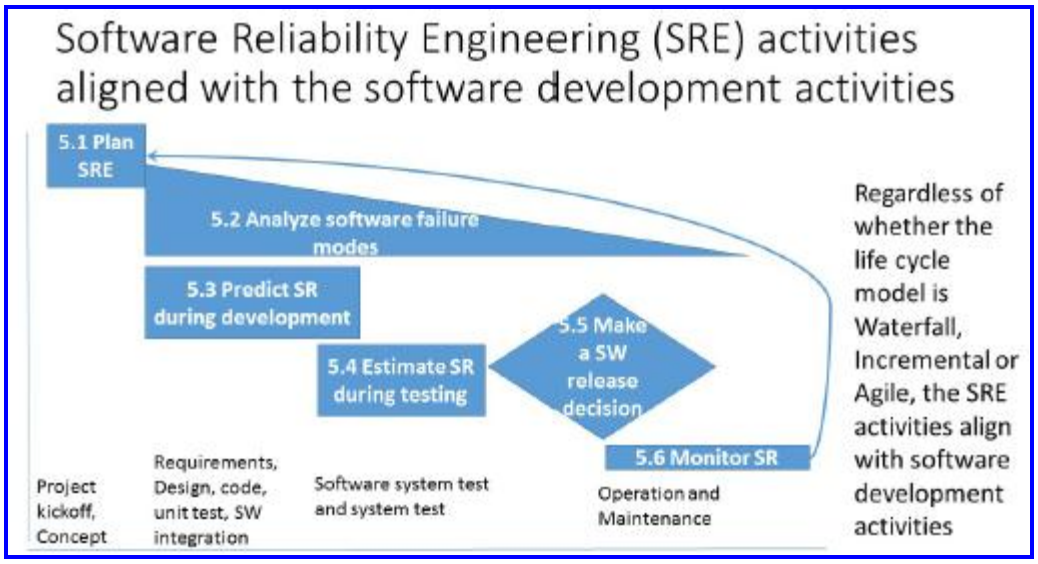
d) **Testing**—Conduct testing that is representative of field usage while also covering the requirements, design, and code so that the software is tested as per its OP. Develop test suites that achieve representative sampling of the operations, modes, and stresses. Count the number of defects discovered (each event is a fault) and corrected, and monitor the discovery rate. As needed revise the predictions. Estimate the system's operational reliability from the frequency and trend of observed failures using a reliability growth model or failure intensity graph. Determine if the estimated number of residual defects is acceptable.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

e) **Release**—Use SRE to support a release decision. Analyze trends in faults and failures to support transitioning from software milestone decisions. Report the release readiness of the software based on its estimated operational reliability and estimated number of residual defects.

f) **Operation**—Monitor and model the number and rate of software failures so as to improve upon future SR analyses.

Each process area is presented in Clause 5. The SR procedures shown in Clause 5 can be applied regardless of the life-cycle model. The tasks are performed whenever the associated development activity is performed unless specified otherwise. The clauses of this document are aligned with the six activities shown in Figure 2.

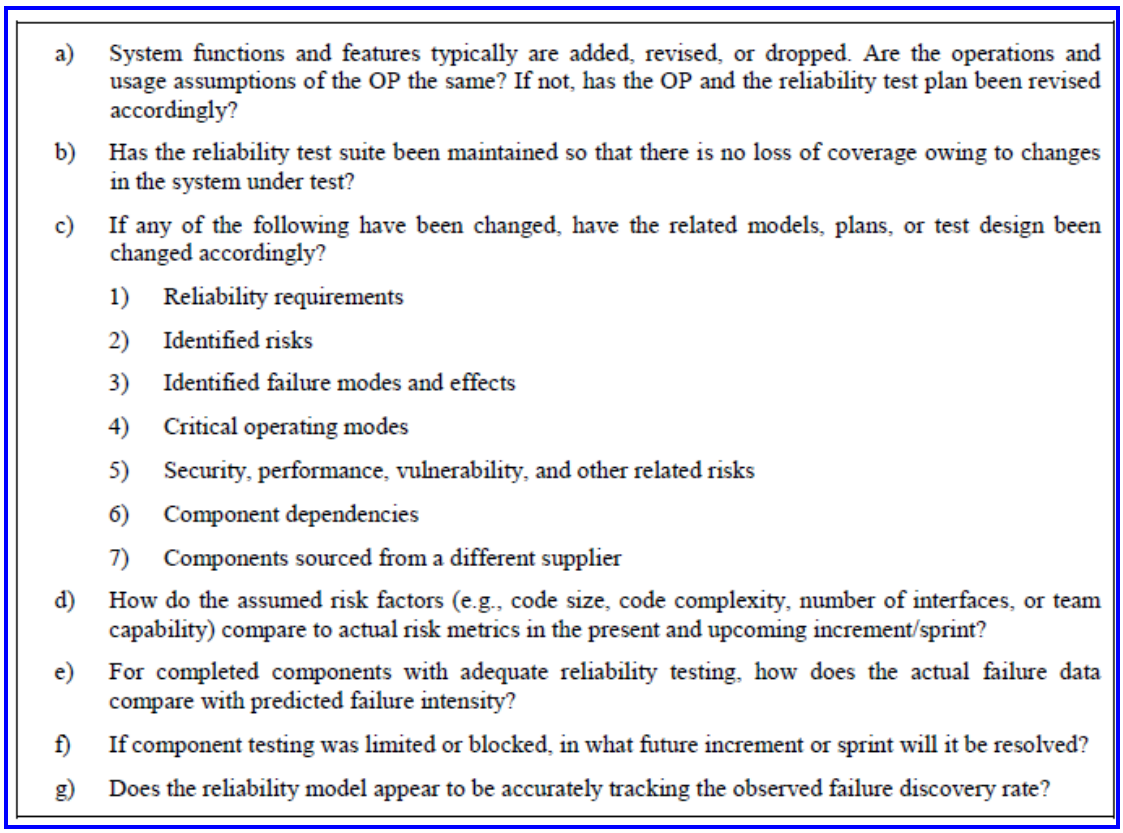


(New)

Figure 2 —Summary of the SRE activities

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

All life-cycle models consist of a sequence of general activities, known by many names, including *phase*, *increment*, and *sprint*. For simplicity, they are called *stages*. As a project progresses through its stages, its assumptions, plans, and commitments change. Therefore, the Software Reliability Program Plan (SRPP) and its activities should be reconsidered and revised or elaborated to maintain alignment and completeness. These transition points occur least frequently in a phased project, more frequently in an incremental project, and most frequently in an evolutionary project. At each stage transition, it is critical to consider what has changed and what these changes imply for all aspects of the reliability plan. Figure 3 is a stage transition checklist.



a) System functions and features typically are added, revised, or dropped. Are the operations and usage assumptions of the OP the same? If not, has the OP and the reliability test plan been revised accordingly?

b) Has the reliability test suite been maintained so that there is no loss of coverage owing to changes in the system under test?

c) If any of the following have been changed, have the related models, plans, or test design been changed accordingly?

   1) Reliability requirements
   2) Identified risks
   3) Identified failure modes and effects
   4) Critical operating modes
   5) Security, performance, vulnerability, and other related risks
   6) Component dependencies
   7) Components sourced from a different supplier

d) How do the assumed risk factors (e.g., code size, code complexity, number of interfaces, or team capability) compare to actual risk metrics in the present and upcoming increment/sprint?

e) For completed components with adequate reliability testing, how does the actual failure data compare with predicted failure intensity?

f) If component testing was limited or blocked, in what future increment or sprint will it be resolved?

g) Does the reliability model appear to be accurately tracking the observed failure discovery rate?

(New)

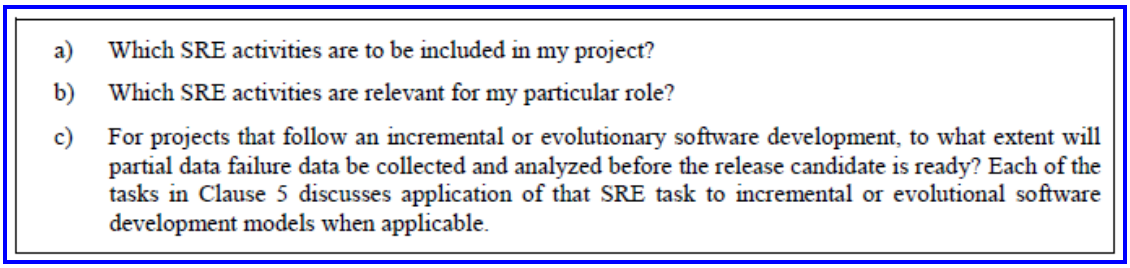
Figure 3 —Stage transition checklist

The questions should be expanded to include any project-specific factors that have a direct bearing on the outcome of the SRE Plan.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

4.3 Project-specific SRE tailoring

The purpose of SRE is to assure that released software meets its reliability requirements using both data-driven quantitative modeling and analysis as well as qualitative analyses. All of the practices in this recommended practice support that goal. Some of them are essential, some are typically useful, and some are project specific. SRE tailoring is the process of deciding which recommended practices to apply to a specific project. Following the tailoring process defined in IEEE Std 12207™-2008, Annex A, is recommended.[8] Taking these general conditions into consideration, the practitioner should answer the following questions in Figure 4.



(New)

Figure 4 —SRE tailoring checklist

Table 1 lists the recommended practices defined in Clause 5 of this recommended practice, indicating the tailorability of each, as follows:

— **Essential** activities should be performed to achieve the basic goals of SRE. If an essential activity is omitted, the risk of an incorrect prediction and inadequate operational reliability is significant.

— **Typical** activities are usually performed to support essential activities and should be included unless there is a good reason to omit them. For example, in a system with just a single software component, multi-component allocation is not necessary.

[8] Information on references can be found in Clause 2.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

— **Project specific** activities are usually performed only when a specific condition warrants their inclusion. For example, in a non-critical system, developing and testing with a separate OP of critical operations is not needed.

Table 1 also illustrates the SRE activities based on the role of the personnel who will typically perform or assist with the activity. Organizations vary with regard to engineering roles, hence the following are the typical roles and responsibilities.

— **Reliability engineers**—These engineers typically have a background in hardware reliability but not necessarily a background in software engineering or SR. Their role is typically to do predictions and merge the predictions into the reliability block diagram (RBD) to yield a system reliability prediction. They also perform allocations that include both software and hardware.

— **Software management**—These engineers manage the day-to-day development activities, which also include scheduling of personnel. Software management uses SR predictions to predict the effort required to maintain the software, schedule the spacing of the releases so as to avoid "defect pileup," schedule the testing and corrective action resources necessary to determine that the software meets the required objective, and perform a sensitivity analysis to determine the development practices that are most and least related to reliable software.

— **Software quality assurance or testing**—These engineers are responsible for audits and assessments of the software deliverables as well as testing of the software. They are typically the persons who use the SR growth models during testing to determine how many more defects are remaining and the test effort required to discover them.

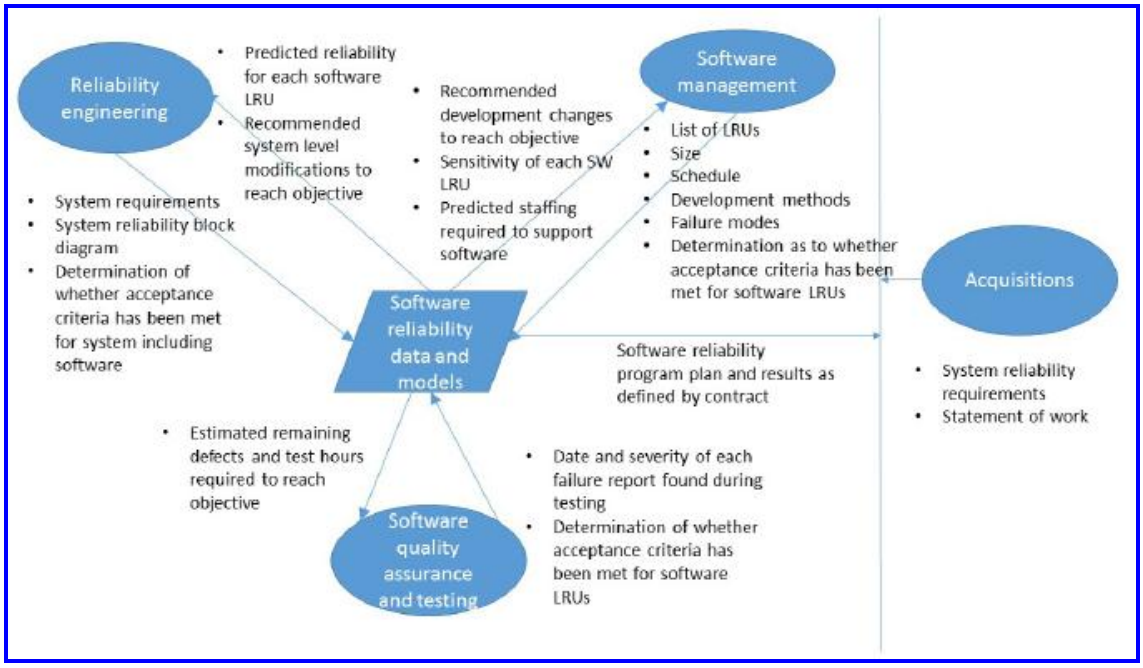
— **Acquisitions**—Acquisitions personnel can be either commercial or government employees. They are purchasing or acquiring software components or an entire system containing software. They may use SR assessments to select or evaluate subcontractors, contractors, or suppliers. They may also use the predictions to establish a system reliability objective to be used for contractual purposes. They also use the SR models to monitor the progress of the software and system reliability.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Reliability engineering and software engineering historically have not had defined relationships. Hence, a SR liaison may be defined to coordinate the efforts of the software organization with the reliability engineering organization. The SR liaison may be a reliability engineer who is knowledgeable about software or a software engineer who is knowledgeable about reliability engineering or a system engineer who is knowledgeable about both.

Typically the SR work is performed at the beginning of a program and at every milestone during the program. It is important that the reliability engineers schedule their efforts to work with all involved parties in preparation for those milestones. Depending on the life-cycle model (LCM), the milestones can vary, hence in 4.4, the SRE effort is illustrated for several different software LCMs.

The following figure illustrates a summary of the role of the software managers, reliability engineers, software quality engineers, and acquisitions. Clause 5 describes the data flow in more detail. Figure 5 illustrates some of the key data that is input to the reliability models and data as well as the key metrics that are retrieved from the SRE models and data.



(New)

Figure 5 —Relationship of SRE stakeholders

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Table 1 illustrates the tasks typically performed by each role. The key to the table is as follows:

Role

RE—Reliability engineer or systems engineer

SQA—Software quality assurance or testing

SM—Software management

ACQ—Acquisitions personnel who are purchasing or acquiring software from other organization

Tailoring

E—Essential

T—Typical

PS—Project specific

Activity for each role

L—This person leads the activity

X—This person provides inputs to the activity

C—This person consumes the results of the activity

R—This person reviews the results of the activity

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Table 1 —Essential, typical, and optional SRE activities by role

Reference	Topic	Tailoring	RE	SQA	SM	ACQ	Related standards
5.1	Planning for software reliability—These are SRE activities that need to be performed prior to using any SRE models or analyses						
5.1.1	Characterize the software system – Identify the software and firmware LRUs, bill of materials (BOMs), OP, and impact of software on the system design.	E	L C		X	R	Software and Systems Engineering Vocabulary (SEVOCAB): http://pascal.computer.org/sev_display/index.action IEEE Std 12207-2008 IEC 61014:2003 [B29]
5.1.2	Define failures and criticality—Every product and system has a unique definition of failure criticality that should be defined in advance so that the software is designed, tested, and measured correctly.	E	C	C	C	L or R	This step may be implemented within the measurement process in 6.3.7 of IEEE Std 12207-2008
5.1.3	Perform a reliability risk assessment—Safety, security, vulnerability, product maturity, project and schedule risks can have an impact on the required reliability.	T	L	X	X or L	L or R	IEEE Std 730™-2014 [B32] Tables C.8 and C.16 (see the definition of integrity level) IEEE Std 15026™-3-2013 [B34]
5.1.4	Assess the data collection system—The analyses and models will require software defect, product and process related data.	T	X	L	X	R	IEEE Std 730™-2014 [B32] Tables 23 and 31, and C 3.6.2
5.1.5	Review available tools needed for SR	T	L	X	X	R	
5.1.6	Develop a *software reliability plan*—Identify what SR activities are relevant for a particular software release.	E	L X	X	X	R	
5.2	Develop failure modes model						IEC 61014:2003 [B29]
5.2.1	Perform software defect root cause analysis (RCA)—Identify the most common type of software defect so that practices can be put in place to identify and remove those categories of defects prior to deployment.	PS	C	L	X	R	PS
5.2.2	Perform software failure modes effects analysis (SFMEA)—This bottom-up analysis is similar to a hardware (HW) FMEA except for the software failure modes and root causes and the software viewpoints.	PS	L	C	X C	L or R	PS
5.2.3	Include software in the system fault tree analysis (FTA)—The FTA is useful for identifying single and multiple point failures from a top-down viewpoint.	PS	L	C	X C	L or R	PS

(New)

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Table 1—Essential, typical, and optional SRE activities by role

(continued)

Reference	Topic	Tailoring	RE	SQA	SM	ACQ	Related standards
5.3	Apply SR during development—All of these activities can be performed well before the code is written or tested.						IEC 61014:2003 [B29] IEEE P24748-5 [B30]
5.3.1	Identify/obtain the initial system reliability objective—The required or desired MTBF, failure rate, availability, reliability for the entire system.	E	L or R	C	C	L or R	
5.3.2	Perform a SR assessment and prediction—Predict the MTBF, failure rate, availability, reliability, and confidence bounds for each software LRU.	E	L C	X	X	L or R	
5.3.3	Sanity check the prediction—Compare the prediction to established ranges for similar software LRUs	T	L			L	
5.3.4	Merge the predictions into the overall system predictions—Various methods exist to combine the software and hardware predictions into one system prediction.	E	L			R	
5.3.5	Determine an appropriate overall SR requirement —Now that the system prediction is complete, revisit the initial reliability objective and modify as needed.	E	L	X	X C	X R	
5.3.6	Plan the reliability growth—Compute the software predictions during and after a specific level of reliability growth testing.	E	L C	X C	X C	R	
5.3.7	Perform a sensitivity analysis—Identify the practices, processes, techniques, risks that are most sensitive to the predictions. Perform trade-offs.	PS	X		L C		
5.3.8	Allocate the required reliability to the software LRUs—The software and hardware components are allocated a portion of the system objective based on the predicted values for each LRU.	T	L		X C	R	See NOTE 2.
5.3.9	Employ SR metrics for transition to testing—These metrics determine if the software is stable enough to be tested.	T	X	L C	X C	R	Software and Systems Engineering Vocabulary (SEVOCAB): http://pascal.computer.org/sev_display/index.action

(New)

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Table 1—Essential, typical, and optional SRE activities by role

(continued)

Reference	Topic	Tailoring	RE	SQA	SM	ACQ	Related standards
5.4	Apply SR during testing—These SRE activities are used once the software LRUS have been integrated.						ISO/IEC 29119-1:2013, Concepts and Definitions (published September 2013); ISO/IEC 29119-2: Test Processes (published September 2013); ISO/IEC 29119-3: Test Documentation (published September 2013); ISO/IEC 29119-4: Test Techniques (at DIS stage, anticipating publication in late 2014)
5.4.1	Develop a reliability test suite	T	L	X	X	R	
5.4.2	Increase test effectiveness via software fault insertion—This is when identified failure modes are instrumented such that the software identifies and recovers from them as required.	PS		C	L C		
5.4.3	Measure test coverage—Measures the software coverage from both a structural and functional standpoint.	T	C	X C	X C	R	
5.4.4	Collect fault and failure data—This data is collected during testing and is required for using the SR metrics and models. This subclause includes two very simple metrics that help determine the best reliability growth model.	E		L	X	R	
5.4.5	Select reliability growth models—The best models are those that fit the observed fault trends.	E	L	X	C	R	
5.4.6	Apply SR metrics—These metrics are used in conjunction with the SR models.	T	C	L X	X	R	
5.4.7	Determine the accuracy of the prediction and reliability growth models—The prediction models in 5.3.2 and the SRG models in 5.4.5 can be validated against the actual observed MTBF and failure rates to determine which model(s) are the most accurate for this particular software release.	E	L	L		R	
5.4.8	Revisit the defect RCA—The defect RCA can be updated once testing is complete and used to improve the defect removal of the next software release.	PS	C	L	X	R	IEEE Std 730™-2014 [B32] Tables C.6 and C.16

(New)

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Table 1—Essential, typical, and optional SRE activities by role

(continued)

Reference	Topic	Tailoring	RE	SQA	SM	ACQ	Related standards
5.5	Support release decision—SR results can be used to determine if the software is ready for release. If it is not, the models can determine how much more testing time and defects are required to be removed to be ready for release.						IEEE Std 1012-2012 [B33]
5.5.1	Determine release stability	E	C	L	C	R	IEEE Std 1012-2012 [B33], 7.2.2, particularly 7.2.2.3.6. See NOTE 3.
5.5.2	Forecast additional test duration—If the required reliability is not met, determine how many more testing hours are required to meet it.	T	C	L	C		
5.5.3	Forecast remaining defects and effort required to correct them—If the required reliability is not met, determine how many more defects and staffing is required to meet it.	E	C	L	C		
5.5.4	Perform a Reliability Demonstration Test (RDT)—Statistically determine whether a specific reliability objective is met.	PS	L			R	
5.6	Apply SR in operation—All of these activities are employed once the software is deployed.						
5.6.1	Employ SR metrics to monitor operational SR—It is important to monitor operational reliability to know how the reliability in the field compares to the estimated and predicted.	T	X C	X	X		
5.6.2	Compare operational reliability to predicted reliability.	T	L			R	
5.6.3	Assess changes to previous characterizations or analyses—Adjust the reliability models inputs and assumptions accordingly to improve the next prediction.	T	X C	X	X		
5.6.4	Archive operational data—Organize the data for use for future predictions.	T	X C	X	X	X	
6.2, Annex B	Methods for predicting defect density and fault profile.	E	X	L		R	
6.3, Annex C	SR growth models	E	C	L		R	

NOTE 1—Within the framework of the standards harmonized with the IEEE Std 12207-2008, the concept of reliability is often bundled with the concept of integrity and expanded in the discussion of integrity levels.

NOTE 2—This step may be implemented within the software requirements analysis process in 7.1.2 of IEEE Std 12207-2008.

NOTE 3—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. See also details for verification and validation processes in IEEE Std 1012-2012 [B33].

(New)

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Annex D contains a relative ranking of the culture change, effort, calendar time, and automation of each of the preceding tasks. Several of the preceding tasks can be merged with and sometimes even replace an already existing software development or reliability engineering task.

4.4 Life-cycle considerations for SRE

The general SRE activities of planning, analysis, testing, and acceptance may be applied to any type of software product and in any kind of LCM. Although the underlying practices are independent of a process model, considerations for application of SRE practices varies among these processes.

IEEE Std 12207-2008 recognizes three basic software LCMs, as follows:

a) *Phased*: Often called "waterfall" development, in a phased project all requirements are defined first. Then design, programming, and testing follow in sequence, with backtracking as necessary. US DoD standard 2167-A provides an example.

b) *Incremental*: Most requirements are defined first, then partitioned into subsets. Requirement subsets are designed, programmed, and tested, typically in a sequence of subsets ("increments.") Later increments typically extend or revise the requirements and artifacts of earlier increments. The Rational Unified Process is a widely used incremental process (see [B74]).

c) *Evolutionary*: Often called "agile" development, an evolutionary project begins with a general notion of a solution. This is divided into subsets to be explored and implemented in short cycles, typically spanning about one month. In each such cycle, usage examples are identified and expressed as test conditions, which are prerequisite for programming. Other work products antecedent to delivered software (e.g., line-item requirements, design documentation) are eschewed in favor of direct interaction with system users or customers. Programming and unit testing is conducted to implement the usage examples selected for a cycle. There are many variations.

In practice, the preceding incremental and evolutionary LCMs are adapted and refined to meet practical considerations.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Being data-driven and working from high-level models to actual implementation measurements, most SRE practices require the results of one or more activities as an input to another. For example, defect predictions require certain codebase metrics and system-scope reliability predictions require either an actual or assumed system structure. As incremental and evolutionary processes produce a series of partial antecedents and implementation, SRE activities should be adjusted accordingly.

The charts in the following subclause show how the software development processes defined in IEEE Std 1220-2008 may be achieved under phased (waterfall), incremental, and evolutionary (agile) development strategies. The IEEE 12207 processes are depicted as horizontal lines. The non-dimensional arrow of time points left to right. The solid area above a line indicates the relative effort allocated to that process. Roughly, the total colored area indicates the total effort expended on the process. The implicit vertical axis for each process is the percent of the process' total effort. Upward and downward ramps suggest increasing and decreasing work. These charts are intended to suggest how SRE practices align with these basic process patterns. The charts are notional and suggestive. They are not intended to be prescriptive. They show that with the necessary changes, SRE may be applied in a wide range of LCMs.

All of the procedures in this document can be used whether there is a waterfall life-cycle model or an incremental life-cycle model. Table 2 summarizes how the procedures are applied for each.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Table 2 —How to apply software reliability for an incremental life cycle

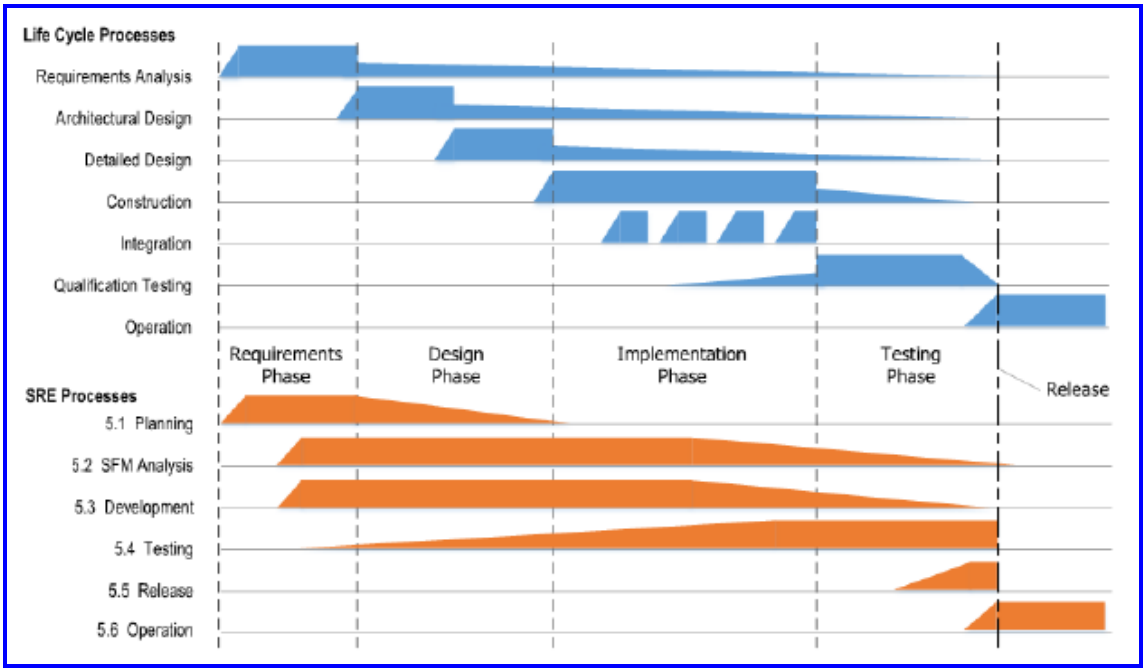
Reference	Phased development life cycle	Incremental and evolutionary development life cycle
5.3.7	Sensitivity analysis using prediction models can identify how to make one release less risky	Sensitivity analysis using prediction models can identify whether moving riskier software features to an earlier increment improves the overall reliability
5.3.9	Metrics are used to determine whether a particular increment should transition to the next phase	Metrics are employed for each increment and also summed for the entire operational release
5.5	Decision making on when to release the software is based on the results of one increment	Decision making on when to release the software is based on the reliability results of the final increment
5.2.3 step b, 5.4.6, 5.5.3	Defect pileup is measured by combining the predicted defects from this release to the actual defect profiles from prior releases as well as future releases.	Defect pileup is measured by merging the predicted defects from each increment with the actual defect profiles from prior releases as well as future releases.
5.3.2.3, 6.2, and Annex B	Prediction models can be used as soon as the concept is defined	Prediction models can be used on entire release or to each internal or external release or both
5.4, 6.3, and Annex C	Reliability growth models can be used as soon as the software is integrated with other software and with the hardware	Reliability growth models can be used for each increment. The results of each increment can be merged for a final reliability growth estimation.

(New)

Figure 6 shows that the processes in a phased life cycle are sequential (a phase may not begin until its antecedent phase is substantially complete) but not mutually exclusive. The long tails indicate that work to accommodate revisions and debugging typically continues for the duration of a project, in most of the processes. The Integration segments assume that partial integration testing is performed as some slice of the system is complete.

This follows common practical use of the phased model, not the naïve assumption of strictly sequenced phases.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016



(New)

Figure 6 —SRE practices in a phased life cycle

Figure 7 is a mapping of the Unified Process Model (UPM) work flows onto the IEEE 12207 processes (see Jacobsen et al. [B36]). The UPM is probably the most widely recognized variant of incremental life cycle. The height and ramps of solid areas closely follow Jacobson et al. Where the UPM defines "implementation" and "test" workflows, these are interpreted as the IEEE 12207 processes "construction," "integration," and "qualification testing."

An incremental project seeks to identify most of the requirements early on. A provisional design is produced and revised in parallel. The requirements and architecture that drive SRE planning, analysis, and prediction are typically substantially complete after the elaboration phase. The system is then constructed in increments, typically with their own integration and end-to-end testing. Revision and elaboration to all work products are expected in the construction phase. As a result, SRE predictions should be revised as new information becomes available. Figure 7 suggests how the focus of SRE activities varies as an incremental project progresses.

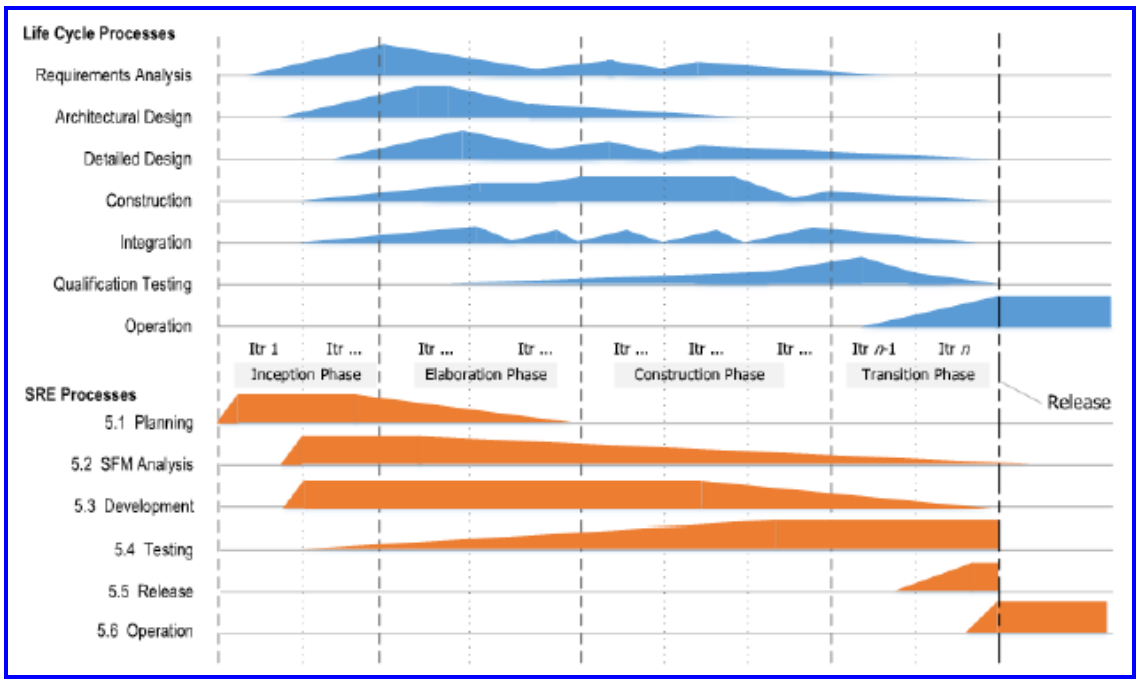
Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Inception—This is a short phase in which the following things are established: justification for the project; project scope and boundary conditions; use cases and key requirements that will drive the design trade-offs; one or more candidate architectures; identified risks; preliminary project schedule and cost estimate.

Elaboration—In this phase the majority of requirements are captured; use cases, conceptual, and package diagrams created.

Construction—The largest phase in which system features are implemented in a series of short, timed iterations.

Transition—In this phase the system is deployed to end users.



(New)

Figure 7 —SRE in an incremental development project

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016

Evolutionary development, generally known as agile, is organized as a sequence of projects to produce small working subsets of a system in a fixed time frame, typically about four weeks. There are many variations of evolutionary development defined by software methodologists. Organizations and project teams have also devised many hybrid process models. In scrum, each subproject is called a "sprint." Typically, the first sprint focuses on overall planning, producing a high-level understand of project scope. Subsequent sprints start with elicitation of a small requirements subset, followed by programming and testing for only that subset. Integration typically occurs throughout a sprint.

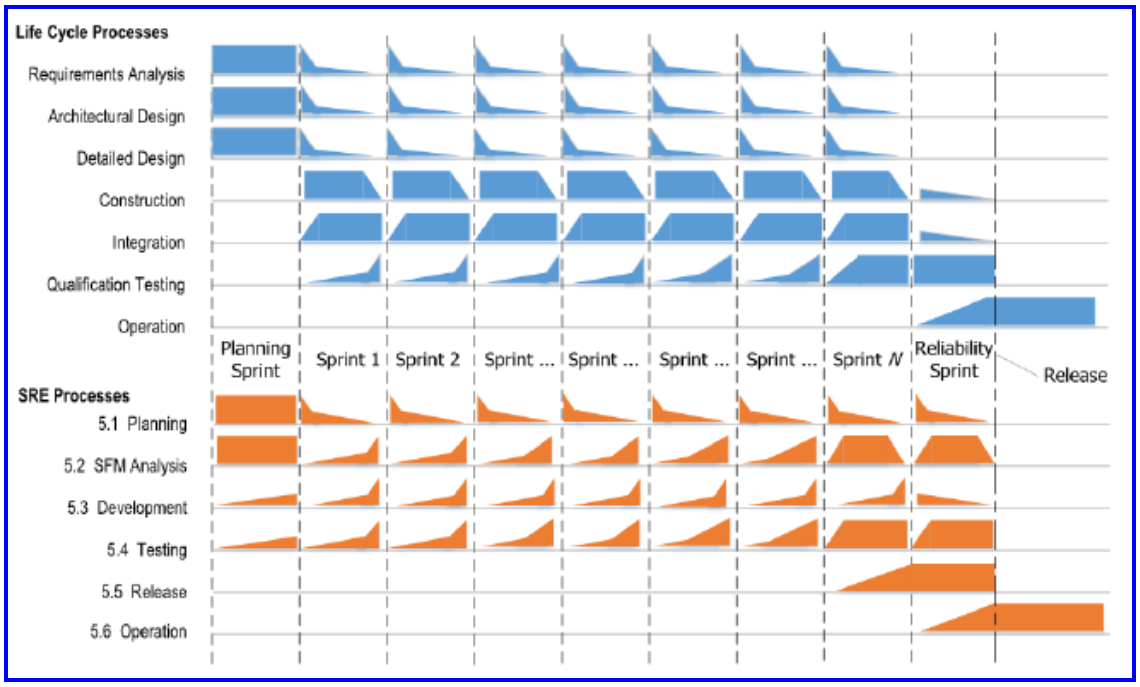
As a result, a system is constructed in many small steps. These steps can produce the software metrics and failure data that drive SRE practices, but until the final sprint, information needed for SRE planning, modeling, and analysis is necessarily incomplete. Refer to Figure 8.

Some evolutionary approaches call for one or more planning sprints and one or more transitional sprints (Ambler and Lines [B2]). A key difference of these sprints is that they are not necessarily intended to produce working software. Planning sprints may be used to initiate SRE planning and analysis. Transitional sprints may be used to apply SRE testing and release evaluation to a release candidate in its entirety. For evolutionary projects, the recommended practice is therefore:

— Include one or more planning sprints at the start of a project to establish a minimal system-scope baseline of SRE planning and analysis artifacts.

— Include one or more reliability sprints at the end of a project to conduct system-scope testing on a completed release candidate so that meaningful reliability predictions may be developed and evaluated. Construction and integration should be limited to making and verifying corrective action fixes, followed by rerunning the system-scope reliability test suite.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016



(New)

Figure 8 —SRE in an evolutionary process

Many software projects may be characterized as maintenance; they are intended to change an existing codebase for perfective, corrective, or adaptive purposes. Even if the reliability requirements of the resulting system are unchanged, the system should be retested with a reliability test suite. During maintenance, SRE is also useful for determining warrantees and determining the number of software engineers who are needed to support the field. Refer to Figure 9.

Documents
Document Sharing Hub
Authorized licensed use limited to: Tsinghua University. Downloaded on March 05,2026 at 17:06:21 UTC from IEEE Xplore. Restrictions apply.
Full document details: IEEE 1633-2016