

	SURFACE VEHICLE/ AEROSPACE RECOMMENDED PRACTICE	SAE JA1003 MAY2012
		Issued 2004-01 Reaffirmed 2012-05 Superseding JA1003 JAN2004
Software Reliability Program Implementation Guide		

RATIONALE

JA1003 has been reaffirmed to comply with the SAE five-year review policy.

Foreword

In 1994, the SAE G-11 Reliability, Maintainability, Supportability and Logistics (RMSL) Division chartered a software committee, G-11SW, to create several software standards and guidance documents across the RMSL spectrum, including a software reliability program standard and implementation guidelines. The committee was formed as a cross section of international representatives from commercial industries and governments.

The G-11SW committee has developed a standard (JA1002) and these implementation guidelines (JA1003) that are consistent with a SAE G-11 system level reliability program standard (JA1000) and guidelines (JA1000-1), augmented by necessary software-specific information. The G-11SW committee believes these documents reflect the best current commercial practices, and meet the objectives of the United States Department of Defense Acquisition Reform initiative and the North Atlantic Treaty Organization (NATO) Reliability Program. The JA1002 program standard is intended to be used by industries to address market demands for reliable software products that improve system productivity, time to market, and cost-effective implementation. The JA1003 guidelines address possible task activities, methods and techniques through which JA1002 can be implemented. As appropriate, governments and other organizations may also reference these documents.

Software has been recognized by SAE G-11 as an important system component that is not adequately addressed at the system level. Software requires interpretation and variations on RMSL methods used by hardware. In particular, the distinction between hardware and software methods may be difficult to define with such products as programmable logic devices and Field Programmable Gate Arrays. This document provides a selection of methods and techniques to implement the simple concept of supplier-customer-certification authority dialogue and partnership to define, meet, and demonstrate assurance of software product reliability requirements. JA1003 describes how to structure a Plan in terms of activities, tasks, and methods so as to achieve the requirements of JA1002 and provide demonstrated Case evidence of reliability achievement.

Development of these guidelines has required dedication by a few participants and extended review by a wider audience of potential users over an extended period of time. The professionalism of all these individuals and the support they received from their companies, governments, and other organizations is gratefully acknowledged.

SAE Technical Standards Board Rules provide that: "This report is published by SAE to advance the state of technical and engineering sciences. The use of this report is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom, is the sole responsibility of the user."

SAE reviews each technical report at least every five years at which time it may be revised, reaffirmed, stabilized, or cancelled. SAE invites your written comments and suggestions.

Copyright © 2012 SAE International

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

TO PLACE A DOCUMENT ORDER: Tel: 877-606-7323 (inside USA and Canada)
 Tel: +1 724-776-4970 (outside USA)
 Fax: 724-776-0790
 Email: CustomerService@sae.org
 http://www.sae.org

SAE WEB ADDRESS:

Full document details: SAE JA1003_201205

SAE values your input. To provide feedback on this Technical Report, please visit
http://www.sae.org/technical/standards/JA1003_201205

Documents

Document Sharing Hub

Abstract

This document defines practices for the implementation of a reliability program for software within an overall systems engineering framework. Guidelines for implementation of a Software Reliability Plan and associated Software Reliability Case are presented. Practices are described for establishing a software reliability program through selection of life cycle activities tailored for the application. Numerous analysis, design, and verification methods and techniques that might be selected to achieve the life cycle activities are summarized and references provided. Guidelines for tailoring a software reliability program include safety and security concerns, integration of Off-The-Shelf software, and collection of appropriate data. The guidelines are applicable to all projects incorporating software, particularly high consequence systems where software reliability is critical. Stakeholders include end-users as well as organizations that acquire, develop or provide post-delivery operation of or support for software.

TABLE OF CONTENTS

1	Scope	6
1.1	Purpose	6
1.2	Audience	6
1.3	Applications	6
1.4	Background	7
1.5	Roadmap to Document Guidance	8
2	References	10
2.1	Applicable Publications	10
2.1.1	SAE Publications	10
2.2	Related Standards	10
2.3	Publications	14
3	Definitions	15
3.1	Acronyms	15
3.2	Terms	16
4	Life Cycle Management	24
4.1	Program Management	24
4.2	Technical Activities	26
4.3	Roles and Responsibilities	30
4.3.1	Customer	31
4.3.2	Supplier/Contractor	31
4.3.3	Certification/Acceptance Authority	31
5	Task Activities	32
5.1	Reliability Analysis Tasks	33
5.1.1	Determine Customer Requirements	34
5.1.2	Meet Customer Requirements	36
5.1.3	Demonstrate Customer Requirements	39
5.2	Reliability Case Documentation	41
5.2.1	System Context Description	42
5.2.2	Goals, Objectives, and Requirements	42
5.2.3	Assumptions and Claims	43
5.2.4	Evidence	43
5.2.5	Conclusions/Recommendations	45
5.2.6	Certification Records	45
6	Special Considerations	46
6.1	Tailoring the Software Reliability Program	46
6.1.1	Application Variations and Complexity	47

6.1.2	Criticality Levels	48
6.1.3	Technique Selection.....	50
6.1.4	Documentation	51
6.2	Safety and Security Considerations.....	52
6.2.1	Safety Considerations	53
6.2.2	Security Considerations	55
6.3	Off-the-Shelf Software and Reuse	58
6.3.1	Support Tools	60
6.3.2	Used in New or Modified Systems	60
6.3.3	During In-Service	61
6.3.4	Documentation	61
6.4	Data Collection and Repositories.....	62
6.4.1	Organizational Responsibilities.....	62
6.4.2	Data Collection	62
6.4.3	Measurement and Metrics	63
6.4.4	Failure Incident Report Form	64
6.4.5	Data Repositories.....	64
Appendix A	Relationship to Existing Standards and Guidelines	68
Appendix B	Example Plan and Case Outlines	70
B.1	Software Reliability Plan Thematic Outline.....	70
B.2	Software Reliability Case Thematic Outline.....	71
Appendix C	Task Activities, Methods, and Techniques	73
C.1	Analysis Techniques	76
C.1.1	Change impact analysis	76
C.1.2	Common Cause Failure Analysis.....	77
C.1.3	Defect Removal Efficiency	77
C.1.4	Design of Experiment (DOE)	79
C.1.5	Formal Scenerio Analysis	81
C.1.6	Goal, Question, Metric (GQM)	82
C.1.7	Hazard Analysis	83
C.1.8	Pareto Analysis	85
C.1.9	Probabilistic Methods	86
C.1.10	Pugh Selection	88
C.1.11	Quality Function Deployment (QFD).....	89
C.1.12	Reliability Allocation	90
C.1.13	Reliability Block Diagrams	92
C.1.14	Reliability Prediction Modeling	94
C.1.15	Response Time, Memory, Constraint Analysis.....	96
C.1.16	Six Sigma	97
C.1.17	Sneak Circuit Analysis	98
C.1.18	Software Failure Modes, Effects, and Criticality Analysis (SFMECA)	98
C.1.19	Software Fault Tree Analysis (SFTA)	100
C.1.20	Software Reliability Engineering (SRE-Musa)	102
C.1.21	Statistical Analysis	103
C.2	Design Techniques	105
C.2.1	Design by Contract.....	105
C.2.2	Fault Tolerant Design.....	106
C.2.3	Formal Methods/Languages	108
C.2.4	Independence, Isolation, Inoperability, Incompatibility (I4)	109
C.2.5	Mistake/Error Proofing	110

C.2.6	Petri Nets.....	112
C.2.7	Software Integrity Checks	114
C.3	Verification Techniques.....	116
C.3.1	Boundary Value Analysis	116
C.3.2	Cleanroom.....	117
C.3.3	Coverage Analysis	118
C.3.4	Dynamic Test Methods	119
C.3.5	Formal In-Process Reviews (Fagan Software Inspections).....	123
C.3.6	Operational Profile	124
C.3.7	Peer Reviews	126
C.3.8	Reliability Bench Marking.....	127
C.3.9	Reliability Estimation Modeling	128
C.3.10	Root Cause Analysis.....	130
C.3.11	Testability Analysis, Fault Inspection, Failure Assertion.....	132
C.4	Management Techniques	133
C.4.1	Configuration Management.....	133
C.4.2	Failure Reporting Analysis and Corrective Action System (FRACAS)	136
C.4.3	Life Cycle Process Standard.....	138
C.4.4	Process Assessment.....	140
C.4.5	Requirements Management.....	144
C.4.6	Risk Management	147
Appendix D	Example: Software Reliability Engineering (SRE)	151
D.1	SRE Process.....	151
D.2	Fone Follower Example	151
D.2.1	List Associated Systems	151
D.2.2	Implement Operational Profiles.....	152
D.2.3	Define “Just Right” Reliability	154
D.2.4	Prepare for Test	154
D.2.5	Execute Test	155
D.2.6	Guide Test.....	155
D.2.7	Collect Field Data	157
D.3	Conclusions.....	157
Appendix E	Example: Software Reliability Program Fragment.....	158
E.1	FAA and DO178B Background and Certification Elements.....	158
E.1.1	Level of FAA Involvement (LOFI).....	159
E.1.2	Means of Compliance	160
E.2	Case Study Background and Objectives	161
E.3	Load Control Software Concept and Initial Assumptions	163
E.4	Determination of Level of FAA Involvement	164
E.5	USIA and IMA Inc Contract.....	165
E.6	Case Study Results.....	165
E.6.1	Strategy	166
E.6.2	Reliability Evidence and Metrics	169
E.6.3	Formal In-Process Review (Software Inspection) Evidence.....	170
E.6.4	System/Integrated Test Evidence	171
E.6.5	Computation of the Predicted and Estimated Reliability.....	172
E.6.6	FAA and NSIA Product Reviews and Development Transition to In-Service.....	174
E.7	Case Study References and Supporting Bibliography	177

LIST OF ILLUSTRATIONS

Figure 1	System/Software Reliability Life Cycle Management	25
Figure 2	Customer-Supplier-Certification Relationship	31
Figure 3	Software Reliability Program Plan-Case Framework	33
Figure 4	Reliability Case: Claims Based on Evidence	42
Figure 5	Layered Approach to Software Reliability Tailoring	46
Figure 6	Reliability Confidence Limits	50
Figure 7	Safety Plan, Case, and Policy Model	54
Figure 8	Example System/Software Safety Life Cycle Process	55
Figure 9	Example Failure Incident Report Form	67
Figure C1	Example Pareto Chart of Software Problems	85
Figure C2	Example Probabilistic Methods Work Flow	87
Figure C3	Example Serial Reliability Block Diagram Computation	93
Figure C4	Example Parallel Reliability Block Diagram Computation	93
Figure C5	Reliability Block Diagram Model of HW/SW Computational Component	94
Figure C6	Elements of a Software FMECA	99
Figure C7	Elements of a Software FTA	101
Figure C8	Example Software Petri Net Diagram (Fragment)	113
Figure C9	Formal In-Process Reviews Across Life Cycle Activities	123
Figure C10	Operational Profile Derivation	125
Figure C11	Elements of a Software Configuration Management System	136
Figure C12	Elements of a System-Software FRACAS	137
Figure C13	Example Integrated Life Cycle Management System	139
Figure C14	Example Software Process Improvement Cycle	141
Figure C15	CMMI Structure and Classes of Assessment Methods	143
Figure C16	Elements of a Requirements Management Framework	145
Figure C17	Example Risk Management Matrix Template	148
Figure D1	SRE Process Steps	152
Figure D2	Plot of FI/FIO Ratio for Fone Follower	156
Figure D3	Reliability Demonstration Chart Applied to Fone Follower	157
Figure E1	Integrated Modular Avionics (IMA) Hardware Element	161
Figure E2	Customer, Supplier, Certification Roles	162
Figure E3	Example USIA Customer and IMA Inc. Supplier Contract Outline	167
Figure E4	Example Evidence: Plot of Defect Data by Life Cycle Activity	173
Figure E5	Example Evidence: Development Transitions to In-Service Support	176
Figure E6	Example Evidence: FRACAS Data Collection for a Sustained Reliability Program During In-Service	177
Table 1	Document Roadmap to Topics of Interest	9
Table 2	Example Defects and Removal Efficiency Profiles	27
Table 3	Example Defect Removal Efficiency, Delivered Defects for Applications	27
Table 4	Practices Associated with Reduced Defect Density	28
Table 5	Example Reliability Failure Measures by Application Type	35
Table 6	Example Format for Summary of Software Reliability Case Evidence	44
Table 7	Example Software Reliability Criticality Matrix	49
Table 8	Major Surety Themes	53
Table 9	Security Principles and Associated Surety Themes	56
Table 10	Example Reliability Data for the AIAA Repository	66
Table A1	Characterization of Standards and Relationship to Software Reliability	69
Table C1	Techniques to Achieve and Assess Software Reliability by Life Cycle Phase	74

Table C2	Defect Origin and Removal Efficiency Metrics.....	78
Table C3	Defect Potential and Removal Efficiency by Quality Level.....	78
Table C4	Defect Density Ranges by SEI CMM Level	78
Table C5	Common Software Reliability Estimation Models	129
Table D1	Fone Follower Operational Profile	153
Table E1	LOFI Scoring Table	159
Table E2	LOFI Categories: High, Medium, Low.....	159
Table E3	DO178B Possible Deliverables.....	160
Table E4	Example Software Reliability Plan Compliance Matrix	168
Table E5	Example Evidence: Formal Inspections.....	171
Table E6	Example Evidence: System/Integration Test Data	171
Table E7	Example Evidence: Inspections, Testing, Defect Removal Efficiency.....	172
Table E8	Example Evidence: Predicted Reliability Calculations.....	173
Table E9	Example Evidence: Estimated Reliability Calculations.....	174
Table E10	Example Evidence: FAA and NSIA Product Reviews.....	175

1. Scope

1.1 Purpose

This document provides methods and techniques for implementing a reliability program throughout the full life cycle of a software product, whether the product is considered as standalone or part of a system. This document is the companion to the Software Reliability Program Standard [JA1002]. The Standard describes the requirements of a software reliability program to define, meet, and demonstrate assurance of software product reliability using a Plan-Case framework and implemented within the context of a system application.

This document has general applicability to all sectors of industry and commerce and to all types of equipment whose functionality is to some degree implemented by software components. It is intended to be guidance for business purposes and should be applied when it provides a value-added basis for the business aspects of development, use, and sustainment of software whose reliability is an important performance parameter. Applicability of specific practices will depend on the reliability-significance of the software, application domain, and life cycle stage of the software.

Following guidelines in this document does not guarantee required reliability will be achieved, or that any certification authority will accept the results as sufficient evidence that requisite reliability has been achieved. Following guidelines in this document will provide insight into what level of reliability has been achieved. With proper customer, certification authority, and supplier negotiation and interaction in accordance with these guidelines, it is more likely that the achieved reliability will be acceptable.

1.2 Audience

The target audience for this document includes customer organizations, certification authorities, specialty reliability engineers, and software developers that acquire, develop, use, or provide post-delivery operation of or support for software.

1.3 Applications

The guidance in this document can be applied to all software-intensive projects, and in particular to projects where the reliability of the software is critical to the performance of the system mission. System applications include military, aerospace, transportation, medical, nuclear industries, ground vehicles, and other consumer applications. Such systems may include the integration of custom software as well as Off-The-Shelf (OTS) software. Custom software is generally newly developed software or a significant rework/upgrade of existing software that is for use with a specific application. OTS software sources include commercial vendors, government, and industry. The guidance in this document is generally applicable throughout the complete life cycle, although specific approaches may be more effectively applied at specific life cycle points depending on the software source, application, and pedigree.

1.4 Background

Software is a major component of most important system applications. Because the software component typically provides critical functions, faults in the software may cause the system to fail in a significant way. Such system failures due to direct cause software faults are what we classify as “software failures”. Thus, it is important to use methods and techniques that provide evidence that the software component has been designed, implemented, tested, installed, and, as necessary, updated without faults that might result in undesirable system failures.

The topic of software reliability is concerned with all life cycle activities that prevent, detect, remove, and/or mitigate software faults, and that verify/validate the degree to which software faults do not exist and will not cause system failures. Software reliability is (quantitatively) defined as the probability of failure-free operation of a software program for a specified time under specified conditions. However, having a “number”, even with the appropriate accompanying evidence, is not generally sufficient to convince customers, regulatory authorities, or even the system/software suppliers that the software satisfies its requirements. Thus, software reliability is also (qualitatively) defined as a set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time. Attributes that relate to implementation of fault tolerance design, use of best practice engineering practices, application of specialized methods and techniques for ensuring safety- and/or security-critical requirements, and procedural methods to ensure mistake-proof loading and/or operation also provide evidence that improves the confidence that the software will not cause a system failure.

There are similarities between hardware and software failures and also differences. Software failures are primarily the result of design defects (during development or maintenance). Other failure sources include use-induced degradation as well as inadequate operational procedures and logistics operations documentation that is considered part of the “software data package”. Hardware failures are primarily the result of physical wear out. Other failure sources include design defects, manufacturing quality deficiency, or maintenance or operating errors. Some system failures are the result of a combination of hardware and software faults. It is generally easier to implement changes to software than to hardware, although any component change must be part of a system support concept that includes continued reliability analysis. Hardware is generally repaired to an original state, unless there is a reason to modify it. Software can frequently be returned to its original state by re-initializing, and often is corrected, enhanced, and adapted so as to become a new version, that is, a new product.

Both hardware and software must be managed as an integrated system. The reliability of the system will depend on the reliability of the hardware and software as an integrated whole. Some techniques to manage the system reliability will be similarly applied to hardware and software components whereas other techniques will be unique to hardware or to software. In addition, the application of a given technique may be different for software than for hardware.

There are no existing methods that guarantee delivered software has no faults. That is, there is always some likelihood that under certain environmental conditions and system operational use, faults in software will be encountered that result in failures of the system. In short, software reliability is not "1.0". There are existing methods and techniques that correlate with delivery of software with reduced faults/failures. It is desirable to provide sufficient quantitative and qualitative evidence that appropriate development and support activities have been conducted to prevent, detect, remove, and/or mitigate possible software faults, particularly those faults that might result in critical system failures.

How might faults be prevented, detected, removed, and/or mitigated in the software development and/or support activities? What techniques might be used to provide quantitative or qualitative evidence that faults capable of causing a system failure do not exist in the software component? Given limited resources and time, which combination of techniques provides the "optimum" cost/benefit results? How are decisions made to select such techniques and how is the evidence from the use of such techniques collected and presented? It is these concerns for which this document provides some guidelines – both for management of a software reliability program and conduct of life cycle activities using appropriate software engineering and reliability-specific techniques.

The reference by Littlewood [LITTLEWD00] describes some of the challenges of providing evidence that can support pre-operational claims for reliability. "Particularly when high levels of reliability need to be assured, it will be necessary to use several sources of evidence to support reliability claims. Combining such disparate evidence to aid decision making is itself a difficult task and a topic of current research." Four areas of evidence are discussed in terms of benefits and limitations:

1. evidence from software components and structure;
2. evidence from static analysis of the software product;
3. evidence from testing of software under operational conditions; and
4. evidence of process quality.

Among the challenges to provide software reliability assurance, there are cultural issues in addition to hard technical research questions to be investigated.

The guidelines in this document recommend determining, meeting, and demonstrating the assurance of customer requirements with an integrated and agreed-upon set of activities and measures within a system context, using a plan-case management framework. It is the hope that these guidelines will be a basis for promoting a systematic approach to the assurance of software reliability through direct attention to the cultural issues of negotiation, implementation agreement, and human interface as well as to the hard technical research necessary to demonstrate progress in understanding this complex area.

1.5 Roadmap to Document Guidance

Each reader of this guide may have different interests. A quick roadmap summary of sections of this guideline that might support reader interests is contained in Table 1.

TABLE 1—DOCUMENT ROADMAP TO TOPICS OF INTEREST

Topic/Question of Interest	Document Section(s) to Read
What are software reliability management concerns across the life cycle?	Section 4
What specific task activities are recommended for a software reliability program?	Section 5
Where is the content of a software reliability plan discussed?	Section 5.1, Appendix B
Where is the content of a software reliability case discussed?	Section 5.2, Appendix B
How do safety and/or security concerns relate to a software reliability program?	Section 6.2
Where are Off-The-Shelf and reused software reliability concerns described?	Section 6.3
What data should be collected as part of a software reliability program?	Section 6.4
Are there any examples that might assist in understanding software reliability?	Appendix D, Appendix E
What analysis, design, and/or verification techniques are available to support software reliability task activities?	Appendix C
What tailoring guidance is provided for a software reliability program?	Section 6.1
What other standards and guidelines exist in this area and how does JA1003 relate to these documents?	Appendix A
What references might be useful to review?	Section 2, Appendix C
Where is the software reliability terminology defined?	Section 3

This guide is not intended to be a novel read from front to back. The life cycle management information described in Section 4 provides an overview of a software reliability program across the various life cycle phases, including example methods/techniques that might support the program in each phase. The task activities described in Section 5 directly support implementation of the software reliability program standard [JA1002] requirements. If the reader is interested in considerations for tailoring a program, addressing safety/security, integrating Off-The-Shelf software, or data collection then Section 6 would be the place to find such information.

The relationship of this software reliability guideline document to many existing standards and guidelines documents is presented as a matrix in Appendix A. Software reliability plan and case outlines are illustrated in Appendix B. If there is interest in a wide variety of potential methods and techniques, then Appendix C would be the place to look. It is emphasized that there are numerous ways to combine and integrate methods and techniques different from those described in Appendix C. There are undoubtedly excellent methods and techniques that exist and are not included in this guide or that are developed after publication of this guide. Numerous tools exist to support the methods and techniques, but this guide does not specifically discuss any of those tools since their capabilities change so rapidly. Such tools can be identified through the many references. Two case studies (at least fragments) are covered in Appendix D and Appendix E. A fairly comprehensive glossary of acronyms and definitions is contained in Section 3 and primary references from SAE, related standards and guidelines, and publications of interest are contained in Section 2. It is also noted that many other references are contained in Appendix C as part of each specific method/technique.

2. References

2.1 Applicable Publications

The indicated references and web links were the current known version as of the publication of this guide. See Appendix C for additional references specific to reliability methods and techniques. There are numerous other publications relevant to software reliability.

2.1.1 SAE PUBLICATIONS

Available from SAE, 400 Commonwealth Drive, Warrendale, PA 15096-0001.

AIR5022—Reliability and Safety Process Integration

ARP5580—Recommended Failure Modes and Effects Analysis (FMEA) Practices for Non-Automobile Applications

J1739—Potential Failure Mode and Effects Analysis in Design (Design FMEA) and Potential Failure Mode and Effects Analysis in Manufacturing and Assembly Processes (Process FMEA) and Effects Analysis for Machinery (Machinery FMEA)

JA1000—Reliability Program Standard

JA1000-1—Reliability Program Implementation Guide

JA1002—Software Reliability Program Standard

JA1004—Software Supportability Program Standard

JA1005—Software Supportability Program Implementation Guidelines

JA1006—Software Support Concept

JA1010—Maintainability Program Standard

JA1010-1—Maintainability Program Implementation Guide

2.2 Related Standards

A world-wide search capability for reliability standards and standards developing organizations is available from: IIT Research Institute / Reliability Analysis Center, 201 Mill Street, Rome, NY 13440-6916

2.2.1 AIAA PUBLICATIONS

Available from American Institute of Aeronautics and Astronautics (AIAA), 1801 Alexander Bell Drive, Suite 500, Reston, VA 20191-4344.

AIAAR013—ANSI/AIAA R-013-1992, "AIAA Recommended Practice for Software Reliability," February 1993.

2.2.2 BRITISH STANDARDS INSTITUTE PUBLICATIONS

Available from British Standards Institute (BSI), Linford Wood Milton Keyes, MK14 6LE UK.

BS5760-P8—BS 5760, "Reliability of Systems, Equipment and Components," Part 8: "Guide to Assessment of Reliability of Systems Containing Software," British Standards Institute, Draft for Approval for Publication, July 7, 1997.

2.2.3 DoD PUBLICATIONS

Available from Chief, Bibliographic Systems, U.S. Government Printing Office, Sales Management Division (SSMB), Washington, DC 20402.

MILSTD882D—MIL-STD-882D, "Department of Defense Standard Practice for System Safety," Department of Defense, February 10, 2000.

NUREG6421—NUREG/CR-6421, "A Proposed Acceptance Process for Commercial Off-the-Shelf (COTS) Software in Reactor Applications," Office of Nuclear Reactor Regulation, US Regulatory Commission, March 1996.

2.2.4 IEC PUBLICATIONS

Available from International Electrotechnical Commission, 1327 Jones Dr., Ann Arbor, MI, 48105.

IEC61508—ISO/IEC 61508, Edition 1.0: "Functional safety—Safety instrumented systems for the process industry sector - Part 1: Framework, definitions, system, hardware and software requirements," Multi-part standard, International Electrotechnical Commission, 1998.

IEC61511-1—ISO/IEC 61511-1, Edition 1.0: "Functional safety of electrical/electronic/programmable electronic safety-related systems," International Electrotechnical Commission, 2003.

IEC61713—ISO/IEC 61713, Edition 1.0: "Software dependability through the software life-cycle processes - Application guide," International Electrotechnical Commission, June 30, 2000.

IEC61719—ISO/IEC 61719 (Draft): "Guide to measures to be used for the quantitative dependability assessment of software," ISO/IEC/TC56/SC7/WG10/N111, Draft February 11, 2000.

2.2.5 IEEE PUBLICATIONS

Available from IEEE Computer Society, Publications Office, 10662 Los Vaqueros Circle, P. O. Box 3014, Los Alamitos, CA 90720-1264.

IEEE12207-0—IEEE/EIA Std 12207.0-1996, "Software life cycle processes," IEEE Computer Society, March 1998.

IEEE12207-1—IEEE/EIA Std 12207.1-1997, "Software life cycle processes - Life cycle data," IEEE Computer Society, April 1998.

IEEE12207-2—IEEE/EIA Std 12207.2-1997, "Software life cycle processes – Implementation considerations," IEEE Computer Society, April 1998.
IEEE610—IEEE Std-610.12-1990, "IEEE Standard Glossary of Software Engineering Terminology," IEEE Computer Society, September 1990.
IEEE982-1—IEEE Std-982.1-1988, "IEEE Standard Dictionary of Measures to Produce Reliable Software," IEEE Computer Society, June 1988.
IEEE982-2—IEEE Std-982.2-1988, "IEEE Guide for the use of Standard Dictionary of Measures to Produce Reliable Software," IEEE Computer Society, September 1988.
IEEE1028—IEEE Std-1028-1994, "IEEE Standard for Software Reviews," IEEE Computer Society, December 1997.
IEEE1220—IEEE Std-1220-1998, "IEEE Standard for Application and Management of the Systems Engineering Process," IEEE Computer Society, December 1998.
IEEE1228—IEEE Std-1228-1994, "IEEE Standard for Software Safety Plans," IEEE Computer Society, March 1994.
IEEE1413—IEEE Std-1413-1998, "IEEE Standard Methodology for Reliability Prediction and Assessment for Electronic Systems and Equipment," IEEE Reliability Society, December 1998.

2.2.6 ISO PUBLICATIONS

Available from Europe: ILI, Index House, Ascot, Berkshire, SL5 7EU, UK, USA: ILI, 610 Winters Avenue, Paramus, NJ 07652, USA, Germany: ILI, Dietlindenstraße 15, D-80802, Munich, Deutschland, Italy: ILI, Via Guido D'Arezzo, 4 - 20145 Milano, France: ILI, 25 rue de Ponthieu, 75008 Paris, France.

ISO12207—ISO/IEC 12207, "Software Life Cycle Processes," August 1, 1995.
ISO15288—ISO/IEC 15288, "Systems Engineering – System Life Cycle Processes," Edition 1, November 8, 2002.

2.2.7 MISRA PUBLICATIONS

Available from Motor Industry Software Reliability Association (MISRA), Electrical Group, MIRA Ltd, Watling Street, Nuneaton, Warwickshire CV10 0TU UK.

MISRA-VBS—ISO/TR 15497, "Development Guidelines for Vehicle Based Software, the Motor Industry," Motor Industry Software Reliability Association, ISBN 0 9524156 0 7, November 1994.

2.2.8 NATO PUBLICATIONS

Available from Directorate of Standardization, Stan 2, Kentigern House, 65 Brown Street, GLASGOW G2 8EX, United Kingdom

ARMP1—ARMP-1, Edition 3, "NATO Requirements for Reliability and Maintainability," June 2002.
ARMP4—ARMP-4, Edition 2, "Guidance on Writing NATO R&M Requirements Documents," October 2001.
ARMP6—ARMP-6, Edition 1, "Monitoring and Managing In-Service R&M," December 1988.
ARMP7—ARMP-7, Edition 1, "NATO R&M Terminology Applicable to ARMPs," July 2001.
NATO96—NATO (Draft), "COTS Software Acquisition Guidelines and COTS Policy Issues—1st Revision," NATO Communications and Information Systems Agency, January 12, 1996.
NATO97—NATO (Draft), "NATO Guidelines for the Integration of Off-The-Shelf Software," Working Paper AC/322(SC/5)WP/4, NATO C3 Board Information Systems Sub-Committee, June 30, 1997.

2.2.9 NIST PUBLICATIONS

Available from National Institute of Standards and Technology, 100 Bureau Drive, Stop 3460, Gaithersburg, MD 20899-3460.

NIST800-14—NIST 800-14, "Generally Accepted Principles and Practices for Securing Information Technology Systems," National Institute for Standards and Technology, 1996.

NIST800-26—NIST 800-26, "Security Self-Assessment Guide for Information Technology Systems," National Institute for Standards and Technology, 2001.

NIST800-27—NIST 800-27, "Engineering Principles for Information Technology Security (A Baseline for Achieving Security)," National Institute for Standards and Technology, 2001.

2.2.10 RTCA PUBLICATIONS

Available from RTCA, Inc., 1828 L Street, NW, Suite 805, Washington, DC 20036.

DO178B—RTCA/DO-178B/ED-12B, "Software Considerations in Airborne Systems and Equipment," Federal Aviation Administration software standard, RTCA Inc., December 1992.

DO248B—RTCA/DO-248, Final Report for Clarification of DO-178B, "Software Considerations in Airborne Systems and Equipment," Prepared by SC-190, October 12, 2001.

2.2.11 SOFTWARE ENGINEERING INSTITUTE PUBLICATIONS

Available from Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890.

CMMI2000—CMMI-SE/SW-Continuous, V1.02, "CMMI for Systems Engineering/Software Engineering, Version 1.02, Continuous Representation," CMU/SEI-2000-TR-019, November 2000.

CMMI-SE/SW-Staged, V1.02, "CMMI for Systems Engineering/Software Engineering, Version 1.02, Staged Representation," CMU/SEI-2000-TR-018, November 2000.

SPICE98—ISO/IEC 15504:1998: "Software Process Improvement Capability Determination (SPICE)—Software Process Assessment," ISO/IEC/JTC1/SC7/WG10/N111, ISO 1998.

2.2.12 UK MINISTRY OF DEFENCE PUBLICATIONS

Available from UK Defence Standardization, Room 1138, Kentigern House, 65 Brown Street, GLASGOW G2 8EX, United Kingdom.

DEFS0042-2—Defence Standard 00-42 (PART 2)/Issue 1, "Reliability And Maintainability Assurance Guides, Part 2: Software," United Kingdom Ministry of Defence, September 1997.

DEFS0042-3—Defence Standard 00-42 (PART 3)/Issue 1, "Reliability And Maintainability Assurance Guides, Part 3: R&M Case," United Kingdom Ministry of Defence, October 1999.

DEFS0055—Defence Standard 00-55 Issue 2, "Requirements for Safety Related Software in Defence Equipment," Part 1: Requirements, Part 2: Guidance," United Kingdom Ministry of Defence, August 1997.

DEFS0060—Defence Standard 00-60, "Integrated Logistic Support", Issue 2, "Logistic Support Analysis Application to Software Aspects of Systems", Part 3, United Kingdom Ministry of Defence, March 1998.

2.3 Publications

- BASIL02—Basili, Vic, Boehm, Barry, and others, "What We Have Learned About Fighting Defects," Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS™02), IEEE Computer Society, 2002. <http://www.CeBASE.org>
- DACS02—DACS CD, "Software Reliability Source Book," Data and Analysis Center for Software, Rome, NY, 2002. <http://iac.dtic.mil/dacs/>
- FALLA96—Falla, Mike, "Results and Achievements from the DTI/EPSRC R&D Programme in Safety Critical Systems," Edited by Mike Falla, Motor Industry Software Reliability Association, November 1996. <http://www.comp.lancs.ac.uk/computing/resources/scs/>
- HELAN98—Helander, M., Shao, M., and Ohlsson, N. "Planning Models for Software Reliability and Cost," IEEE Transactions on Software Engineering, Vol 24, Number 6, June 1998, pp 420-434.
- HERRM99—Herrmann, D., Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors, IEEE Computer Society, Los Alamitos, CA, 1999.
- JONES97—Jones, Capers, "Software Quality In 1997: What Works and What Doesn't," Software Productivity Research, 1997. <http://www.spr.com/>
- LAKEY97—Lakey, Peter and Neufelder, Ann Marie, "System and Software Reliability Assurance Notebook," Rome Laboratory Report, Griffiss Air Force Base, Rome NY, 1997. <http://www.cs.colostate.edu/~cs530/rh/>
- LEVESON95—Leveson, Nancy G., Safeware: System Safety and Computers, Addison Wesley Publishing Company, 1995.
- LITTLEWD00—Littlewood, B. and Strigini, L., "Software Reliability and Dependability: a Roadmap, in The Future of Software Engineering", State of the Art Reports given at the 22nd Int. Conf. on Software Engineering, Limerick, June 2000, (A. Finkelstein, Ed.), pp. 177-188, ACM Press, 2000.
- LYU96—Lyu, Michael, Handbook of Software Reliability Engineering, ISBN 0-07-039400-8, McGraw Hill, 1996.
- MUSA92—Musa, John D. "Operational Profiles in Software Reliability Engineering," IEEE Software, March 1993, pages 14-32.
- MUSA99—Musa, John D., Software Reliability Engineering, McGraw-Hill Book Company, NY, 1999.
- NEUF02—Neufelder-Owner, A., N., "The Facts About Predicting Software Defects and Reliability," Journal of the RAC, 2ndQ, 2002, pp 1-4.
- PRIM97—PRIM-97, "Worldwide Reliability & Maintainability Standards," Reliability Analysis Center, IIT Research Institute / Reliability Analysis Center, Rome, NY, 1997.
- SCHN97—Schneidewind, N., "Reliability Modeling for Safety-Critical Software," IEEE Transactions on Reliability, Vol 46, Number 1, March 1997, pp 88-98.
- SSSHDBK99—Joint Software System Safety Committee and EIA G-46 Committee, "Software System Safety Handbook," Joint Services Computer Resources Management Group, U.S. Navy, U.S. Army, U.S. Air Force, 1999.
- XTALK03—CrossTalk, "Programming Languages," Journal of Defense Software Engineering, Vol. 16 No. 2, February 2003.

3. Definitions

3.1 Acronyms

AIAA	American Institute of Aeronautics and Astronautics
AIR	Aerospace Information Report
ANSI	American National Standards Institute
ARMP	Allied Reliability and Maintainability Publication
ASIC	Application Specific Integrated Circuit
BSI	British Standards Institute
CM	Configuration Management
CMMI	Capability Maturity Model Integrated
COTS	Commercial Off-The-Shelf
DACS	Data and Analysis Center for Software
DoD	Department of Defense
DOE	Design Of Experiment
DSI	Delivered Source Instructions
EIA	Electronics Industries Alliance
FAA	Federal Aviation Administration
FIR	Formal In-Process Review
FMECA	Failure Modes, Effects and Criticality Analysis
FRACAS	Failure Reporting and Corrective Action System
FTA	Fault Tree Analysis
GQM	Goal, Question, Metric
GUI	Graphical User Interface
HCI	Human Computer Interface
I4	Independence, Isolation, Inoperability, Incompatibility
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers
ISO	International Organization for Standardization
JA	Two character code for SAE ground vehicle (J) and aerospace (A) standards and guidelines
KSLOC	Thousands (K) of Source Lines of Code
MISRA	Motor Industry Software Reliability Association
MOD	Ministry Of Defence (United Kingdom)
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
NCSLOC	Non-Commented Source Lines of Code
NDI	Non-Developmental Item
NIST	National Institute of Standards and Technology
OTS	Off-The-Shelf
QA	Quality Assurance

QFD	Quality Function Deployment
R&M	Reliability and Maintainability
RAC	Reliability Analysis Center
RMSL	Reliability, Maintainability, Supportability, Logistics
SAE	Society of Automotive Engineers
SEI	Software Engineering Institute
SFMECA	Software FMECA
SFTA	Software FTA
SRE	Software Reliability Engineering
UK	United Kingdom
V&V	Verification and Validation

3.2 Terms

The following key terms are defined. Reference [IEEE610] is a generally applicable reference for software terms not defined in this section. Some terms apply only to software, but many terms apply more generally to a system of which software is a component. The specific source of the definitions in this section is referenced as follows:

- [0] term defined by its use in this guide
- [1] reference [AIAAR013]
- [2] reference [JA1002]
- [3] reference [JA1000-1]
- [4] reference [ISO12207]
- [5] reference [MUSA99]
- [6] reference [JA1005]
- [7] reference [DO178B]
- [8] reference [MIL-STD-882D]

3.2.1 ACQUIRER[4]

An organization that procures a system, software product or software service from a supplier.

NOTE—The acquirer could be one or more of the following: buyer, owner, user, and/or purchaser.

3.2.2 CERTIFICATION[7]

Legal recognition by the certification authority that a product, service, organization or person complies with the requirements. Such certification comprises the activity of technically checking the product, service, organization or person and the formal recognition of compliance with the applicable requirements by issue of a certificate, license, approval or other documents as required by national laws and procedures.

3.2.3 CERTIFICATION AUTHORITY/REGULATOR [7]

The organization or person responsible within the state or country concerned with the certification of compliance with the requirements.

3.2.4 CONTRACT[4]

A binding agreement between two parties, especially enforceable by law, or a similar internal agreement wholly within an organization, for the supply of software service or for the supply, development, production, operation, or maintenance of a software product.

3.2.5 COVERAGE[0]

The ratio of actual to possible software features/functions, requirements, statements, and/or branches/paths that are exercised during one or more test cases. Types of coverage can be categorized by the unit, e.g., feature coverage, requirements coverage, statement coverage, path coverage.

3.2.6 CUSTOMER[0]

See Acquirer.

3.2.7 DEFECT

See Software Defect.

3.2.8 DEPENDABILITY[0]

See Surety.

3.2.9 DESIGN RELIABILITY[0]

(1) The set of activities that focus on the prevention, detection, prediction, estimation, and/or mitigation of defects in software specifications (e.g., user guide, requirements, design, code, test plan/cases). (2) A measure of the remaining defects in software specifications at a specific reference point. (3) A measure of the predicted software reliability at a specific reference point.

3.2.10 DEVELOPER[4]

An organization that performs development activities (including requirements analysis, design, testing through acceptance) during the software life cycle process.

3.2.11 ERROR[1]

(1) A discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition. (2) Human action that results in software containing a fault.

3.2.12 FAILURE[1]

(1) The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered and a loss of the expected service to the user results. (2) The termination of the ability of a functional unit to perform its required function. (3) A departure of program operation from program requirements.

3.2.13 FAILURE INTENSITY[5]

See Failure Rate.

3.2.14 FAILURE MODES, EFFECTS AND CRITICALITY ANALYSIS [3]

A proactive approach used for determining the potential failure modes of a system/equipment (including software), all likely ways in which a component or equipment can fail, causes for each failure mode, and effects/criticality of each failure mode.

3.2.15 FAILURE RATE[1]

(1) The ratio of the number of failures of a given category or severity to a given period of time; for example failures per second of execution time, failures per month. Synonymous with failure intensity. (2) The ratio of the number of failures to a per unit of time, failures per number of transactions, failures per number of computer runs.

3.2.16 FAILURE REPORTING AND CORRECTIVE ACTION SYSTEM [3]

A set of processes, procedures, and tools for reporting, reviewing, analyzing, correcting, and storing information about system/software failures.

3.2.17 FAILURE SEVERITY[ADAPTED FROM 1]

A rating system for the impact of every recognized credible failure mode.

3.2.18 FAULT

See Software Fault.

3.2.19 FAULT TOLERANCE[1]

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

3.2.20 FAULT TREE ANALYSIS [3]

An analysis technique where identified potential system failure modes are analyzed in terms of what potential software faults (single point of failure) or multiple faults (multiple points of failure) might result in the potential failure mode.

3.2.21 14- INDEPENDENCE, ISOLATION, INOPERABILITY, INCOMPATIBILITY[0]

3.2.21.1 *Independence*

Multiple, independent subsystems and completely different sources of enabling stimuli for critical functions are incorporated within the system.

3.2.21.2 *Isolation*

Critical functions are encapsulated separate from any other functions that might cause undefined interactions with the critical functions

3.2.21.3 *Inoperability*

Critical functions become predictably and irreversibly inoperable in credible abnormal operating environments before the isolation features are compromised

3.2.21.4 *Incompatibility*

Functional interfaces are constructed so that they are incompatible with functions (in particular safety critical functions) with which they are not intended to interface

3.2.22 LIFE CYCLE MODEL[4]

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.

3.2.23 NON-DELIVERABLE ITEM[4]

Hardware or software product that is not required to be delivered under the contract but may be employed in the development of a software product.

3.2.24 OFF-THE-SHELF PRODUCT[4]

Product that is already developed and available, usable either "as is" or with modification.

3.2.25 OPERATIONAL PROFILE[5]

The complete set of operations (major system logical tasks) with their probabilities of occurrence.

3.2.26 OPERATIONAL RELIABILITY[0]

(1) The set of dynamic test activities that focus on the prevention, detection, prediction, estimation, and/or mitigation of defects in the operational software code through dynamic unit, integration, acceptance, certification testing). (2) A measure of the remaining faults in software code at a specific reference point. (3) A measure of the estimated software reliability at a specific reference point.

3.2.27 PROCESS[4]

A set of interrelated activities, which transform inputs into outputs.

NOTE—The term "activities" covers use of resources.

3.2.28 QUALIFICATION[4]

The process of demonstrating whether an entity is capable of fulfilling specified requirements.

3.2.29 QUALITY ASSURANCE[4]

All the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill requirements for quality.

NOTES—(1) There are both internal and external purposes for quality assurance: (a) Internal quality assurance: within an organization, quality assurance provides confidence to management; (b) External quality assurance: in contractual situations, quality assurance provides confidence to the customer or others. (2) Some quality control and quality assurance actions are interrelated. (3) Unless requirements for quality fully reflect the needs of the user, quality assurance may not provide adequate confidence.

3.2.30 SAFETY[8]

Freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment.

3.2.31 SECURITY[0]

Features and procedures of a system that ensure its requirements are met for timely access to authenticated services and for protection from denial of authenticated services.

3.2.32 SOFTWARE DEFECT[0]

Any condition in a software artifact (e.g., specification, code, test) that if left unchanged could result in a software failure. Defect and fault are sometimes considered to be synonymous although fault is more strictly considered to be a defect in the code.

3.2.33 SOFTWARE FAILURE[2]

The inability of a software component to perform its required functions within specified performance requirements.

3.2.34 SOFTWARE FAULT[1]

(1) A defect in the code that can be the cause of one or more failures. (2) An accidental condition that causes a functional unit to fail to perform its required function. Synonymous with bug.

3.2.35 SOFTWARE FAULT[2]

An accidental condition that causes a software functional unit to fail to perform its required function.

3.2.36 SOFTWARE FAULT DENSITY[0]

The ratio of code faults to a unit of size, such as function points, modules, source lines of code at a specific reference point of time, such as at the start of system test or operational use.

3.2.37 SOFTWARE MAINTAINABILITY[6]

The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. Also, a set of attributes that bear on the effort needed to make specified modifications.

3.2.38 SOFTWARE MAINTENANCE[6]

The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment.

3.2.39 SOFTWARE MODIFICATION SUPPORT[6]

The software support activities of change analysis, implementation, test and release of software products. Changes may be termed corrective, perfective and adaptive, and may also embrace modifications that are designed to prevent foreseeable future software operating problems.

3.2.40 SOFTWARE PRODUCT[1]

The set of computer programs, procedures, and possibly associated documentation and data.

3.2.41 SOFTWARE RELIABILITY[2]

(1) The probability of failure-free operation of a software program for a specified time under specified conditions. (2) A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.

NOTE—Quantitative descriptions of software reliability are typically not sufficient evidence that software is reliable, and other evidence such as fault tolerant design, coverage measures, engineering practices, safety analyses, and so forth are necessary to provide adequate confidence that software can be reliably loaded, operated, and supported. Hence, the combination of evidence provided from both definitions (1) and (2) above is typically used to determine the reliability of software.

3.2.42 SOFTWARE RELIABILITY CASE[0]

The evidence presented throughout the project that software reliability requirements are consistent with system level requirements, are achievable, are understood by the development organization, and that ambiguities have been resolved.

3.2.43 SOFTWARE RELIABILITY ENGINEERING[1]

The application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems.

3.2.44 SOFTWARE RELIABILITY ESTIMATION[1]

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

3.2.45 SOFTWARE RELIABILITY MANAGEMENT[0]

The process of optimizing the reliability of software across the complete software life cycle by emphasizing human error prevention, fault detection and removal, use of measurements to improve reliability, and balancing the level of reliability consistent with project constraints such as resources, schedule, and performance.

3.2.46 SOFTWARE RELIABILITY MODEL[1]

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.

3.2.47 SOFTWARE RELIABILITY PLAN[0]

A description of the set of activities that will be performed throughout a project to ensure that requirements for software reliability have been defined through negotiations with the customer, analyses have been identified and conducted that ensure customer reliability requirements are met, and demonstrated evidence is provided that the customer reliability requirements have been achieved.

3.2.48 SOFTWARE RELIABILITY PREDICTION[1]

A forecast of the reliability of the software based on parameters associated with the software product and its development environment.

3.2.49 SOFTWARE RELIABILITY PROGRAM[0]

The management infrastructure and activities necessary to adequately integrate software reliability within a system reliability program and provide adequate evidence that the software reliability requirements have been determined, met, and demonstrated. The two key components of the management infrastructure are the Software Reliability Plan and Software Reliability Case.

3.2.50 SOFTWARE RELIABILITY PROGRAM [ADAPTED FROM 3]

The organizational processes and practices that are intended to: (1) Ensure the delivery of a software product that has been adequately designed to achieve its performance specifications within its system application context; and (2) Ensure there is adequate evidence that the performance specification for the delivered software product has been achieved and continues to be met during operational use.

3.2.51 SOFTWARE SAFETY[0]

Features and procedures which ensure that a software product performs predictably under normal and abnormal conditions, thereby minimizing the likelihood of an unplanned event occurring, controlling and containing its consequences, and preventing accidental injury, death, destruction of property and/or damage to the environment, whether intentional or unintentional.

3.2.52 SOFTWARE SECURITY[0]

Features and procedures of the software component of a system that ensure the system's requirements are met for timely access to authenticated services and for protection from denial of authenticated services.

3.2.53 SUPPLIER[4]

An organization that enters into a contract with the customer for the supply of a system, software product or software service under the terms of the contract.

NOTES—(1) The term "supplier" is synonymous with contractor, producer, seller, or vendor. (2) The customer may designate a part of its organization as supplier.

3.2.54 SURETY[0]

Attributes of and activities associated with achieving and assessing system safety, security, and reliability.

3.2.55 SYSTEM[1]

(1) A collection of people, machines and methods organized to accomplish a set of specific functions. (2) An integrated whole that is composed of diverse, interacting, specialized structures and subfunctions. (3) A group or subsystem united by some interaction or interdependence, performing many duties but functioning as a single unit.

3.2.56 SYSTEM RELIABILITY[3]

The ability of a system to perform a stated function under stated conditions, for a stated period of time.

3.2.57 SYSTEM SAFETY[8]

The application of engineering and management principles, criteria, and techniques to achieve acceptable mishap risk, within the constraints of operational effectiveness and suitability, time, and cost, throughout all phases of the system life cycle.

3.2.58 TIME[1]

There are several categories of time that may be of interest for determining when failures occur and the impact of the frequency of the failures. These categories include: (1) Calendar Time: chronological time, including time during which a computer may not be running. (2) Clock Time: elapsed wall clock time from the start of program execution to the end of program execution. (3) Execution Time: the amount of actual processor time used in executing a program.

3.2.59 VALIDATION[4]

Confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled.

NOTES—(1) In design and development, validation concerns the process of examining a product to determine conformity with user needs. (2) Validation is normally performed on the final product under defined operating conditions. It may be necessary in earlier stages. (3) "Validated" is used to designate the corresponding status. (4) Multiple validations may be carried out if there are different intended uses.

3.2.60 VERIFICATION[4]

Confirmation by examination and provision of objective evidence that specified requirements have been fulfilled.

NOTES—(1) In design and development, verification concerns the process of examining the result of a given activity to determine conformity with the stated requirement for that activity. (2) "Verified" is used to designate the corresponding status.

4. *Life Cycle Management*

The software reliability program is most effective within a complete life cycle management approach as illustrated in Figure 1. Although it is recommended to initiate a program early in the life cycle, one can be established at any point in the life cycle. It may be more difficult to start a program during later system stages, and although certain practices may not be appropriate there are always some practices that will be effective.

Management and technical activities that might be part of a software reliability program are summarized within this section and specific program tasks are described in Section 5. A comprehensive set of potential analysis, design, and verification methods and techniques that can be applied in various life cycle phases are summarized in Appendix C. Some special software reliability considerations that may need to be integrated into a software reliability program are discussed in Section 6.

4.1 Program Management

The use of a software reliability plan/case framework to capture activity planning and results is the recommended reliability program management mechanism. A software reliability program should be implemented within the context of the operational system in which the software is to operate. In addition, such a program must consider the overlap of requirements that may arise from closely related specialty disciplines, such as safety, security, and supportability, as well as quality engineering in general. Since software reliability is a measure of meeting requirements, this measure should include requirements that derive from all concerns. It is essential that the definition of what constitutes failure is clearly stated and related to requirements.

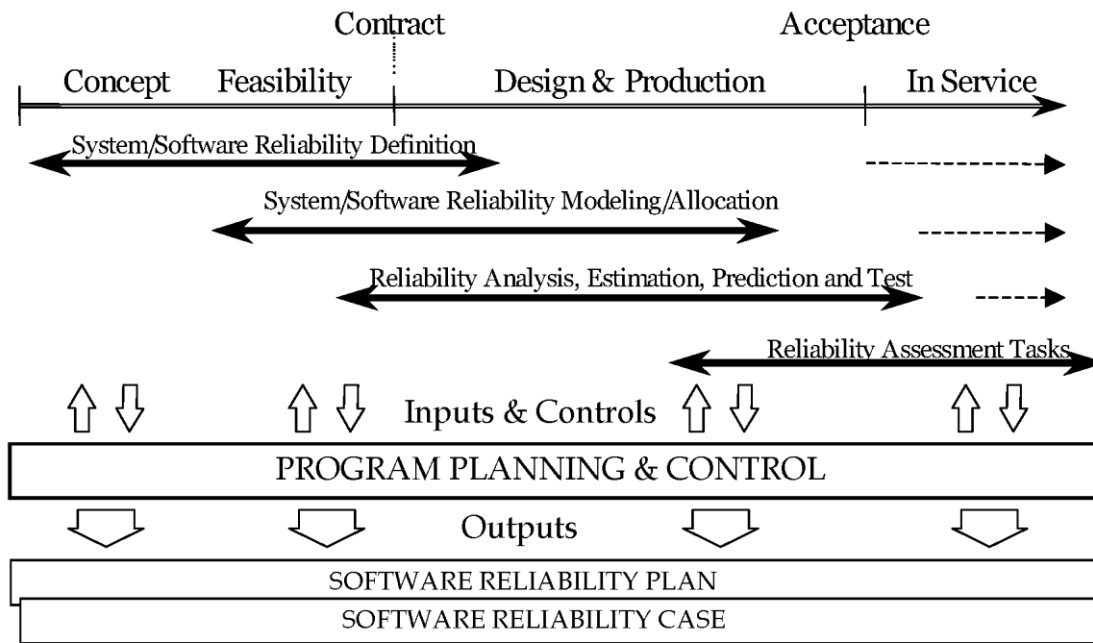


FIGURE 1—SYSTEM/SOFTWARE RELIABILITY LIFE CYCLE MANAGEMENT

Achieving the required reliability for any particular software product will depend on having an effective strategy of planned activities. The strategy should be developed and executed in conjunction with overall system reliability plans. There should be visibility to project and higher program managers of major milestones and outputs to ensure that control and monitoring of reliability activity is integrated with other project activities.

A software reliability plan documents the strategy for satisfying customer reliability requirements and expectations for a software product. This "plan" can assume a variety of physical forms (e.g., separate document, integrated part of other documents, activities integrated into a system Work Breakdown Structure). A software reliability plan needs to address the software aspects of a system reliability plan and describe the specific software-related life cycle activities that are to be undertaken to achieve and sustain the software reliability objectives. Each planned activity should describe associated tasks, schedules, resources, expected methods/techniques to accomplish the activity, and specific metrics to show progress toward goals throughout the life cycle. Results of the activities are described in the software reliability case.

Some of the applicable management practices include the use of life cycle software engineering standards (C.4.3), requirements management (C.4.5), risk management (C.4.6), configuration management (C.4.1), and FRACAS (C.4.2) within the overall system/software project management.

4.2 Technical Activities

Engineering methods can provide confidence that potential faults have been prevented, detected, removed, or mitigated to an acceptable level of confidence. Prototype experiments and trade-offs may be conducted to determine operational profiles and preferred design solutions, depending on the negotiated agreement between the supplier and customer. Software reliability requirements are allocated from system reliability requirements. Software reliability computations are integrated as part of the system reliability computations.

When security, safety, or other high criticality requirements are identified, then specific additional methods can and should be conducted to provide necessary reliability evidence. Depending on the program, this evidence may be separated into distinct "case" reports for reliability, safety, security and so forth, or combined into one "case" report. Thus functional requirements and requirements that derive from specialty interests, as well as selected system- and software-specific methods/techniques determine the context in which a software reliability technical activities are conducted.

In addition, there should be feedback of lessons learned and case evidence from such software reliability activities to enable process improvement. Such feedback should be stored in an organization's overall reliability data base. Thus, each software product's reliability plan and case information can be used in a broader context to support an organization's software reliability process improvement program.

The general criteria for selection of a methods to support technical activities include:

1. substantive use;
2. correlation with technical measurement of defect/failure reduction; and
3. evidence of good benefit to cost performance.

These criteria are typically based on rather qualitative judgment. It is important to understand where defects/faults typically occur in software products as well as the types of defects so specifically effective technical activities can be selected for use. Each application, with customer-/supplier-specific influences, will tend to have unique defect injection/detection/removal profiles. A general view from reference [JONES97] of this type of information for leading, average, and lagging projects is illustrated in Table 2. In addition, defect potential, defect removal efficiency, and delivered defects for a variety of application areas are illustrated in Table 3.

NOTE—Data is in defects per function point.

TABLE 2—EXAMPLE DEFECTS AND REMOVAL EFFICIENCY PROFILES

Task	Leading Organization	Average Organization	Lagging Organization
Requirements	0.55	1.00	1.45
Design	0.75	1.25	1.90
Coding	1.00	1.75	2.35
User manuals	0.40	0.60	0.75
Bad fixes	0.10	0.40	0.85
Total Potential Defect Density	2.80	5.00	7.30
Defect Removal %	95%	85%	75%
Actual Delivered Defect Density (Defects/Function Point)	0.14	0.75	1.83

TABLE 3—EXAMPLE DEFECT REMOVAL EFFICIENCY, DELIVERED DEFECTS FOR APPLICATIONS

	System Software	Commercial Software	Information Software	Military Software	Overall Average
Defect Potentials	6.0	5.0	4.5	7.0	5.6
Defect Removal Efficiency	94%	90%	73%	96%	88%
Delivered Defect Density (Defects/Function Point)	0.4	0.5	1.2	0.3	0.65
First Year Discovery Rate	65%	70%	30%	75%	60%
First Year Defect Density (Defect/Function Point)	0.26	0.35	0.36	0.23	0.30

The following general activities from reference [NEUF02] as illustrated in Table 4 have been correlated with a higher reduction in defects.

TABLE 4—PRACTICES ASSOCIATED WITH REDUCED DEFECT DENSITY

Practice	System Phase & SW Activity
All requirements are mapped to system tests	Concept/Feasibility (SW Requirements)
Requirements are reviewed before designing or coding	Concept/Feasibility (SW Requirements)
System test beds are used	Design&Production (SW Testing)
Test plan started at least one phase of the life cycle before testing begins	Phases before testing (SW Testing)
Testers use a FRACAS (defect tracking system) to determine what to test/retest	Design&Production (SW Testing)
All upgrades made after a system test are regression tested	In Service (SW Support)
Corrective action releases per year ≤ 4	In Service (SW Support)
All modifications made after a system test are regression tested	In Service (SW Support)
FRACAS used for tracking all corrective actions	In Service (SW Support)
Walk-thrus are performed for all phases of life cycle	All phases

The type of information provided in Tables 2, 3, and 4 can provide guidance to a software project for the effective selection of software engineering and reliability techniques.

Some general areas of technical activity and methods to support that activity throughout the typical life cycle activities are summarized below.

a. Negotiate Reliability Requirements

Reliability requirements are negotiated among supplier, customer and certification authority representatives. The requirements for software reliability contribution to overall system reliability is derived from an understanding of the system reliability goals and objectives. An overall strategy for software reliability achievement is determined including pre-development, development, and in-service concepts, activities, methods and tools, risk analysis, and training and integration with other disciplines and staff. Activity-associated measures are established in terms of goals, assumptions and claims, and expected evidence to meet the claims. It is expected that the reliability requirements may continue to be negotiated somewhat throughout the life cycle.

Example Methods: Quality function deployment, customer surveys, Goal-Question-Metric, operational profiles/scenarios, support scenarios.

b. Develop An Architecture/Design for Reliability

Design the software architecture and implementation to support design requirements for software reliability, and iterate with the system and software design effort to ensure reliability characteristics are built into the software artifacts. Ensure the design and implementation meet specified reliability claims through appropriate defect prevention, detection, removal, and mitigation activities.

Example Methods: Software Reliability Engineering, FMEA, FTA, hazard analysis, design by contract, partitioning, formal specifications/analysis, formal scenario analysis/operational profiles, reliability predictions.

c. Construct a Test Approach as Early as Possible

Test planning should be conducted at least one phase prior to test execution. Reliability engineered testing is planned as part of the test planning approach. Testing is critical providing quantitative evidence that reliability requirements have been met. Ensure testing is designed to meet expected reliability claims, including measurement and categorization of defects found and associated measures of reliability growth.

Example Methods: Test plan/harness developed during design; coverage analysis; reliability predictions, reliability estimation.

d. Apply a Formal Review Process During All Phases of Life Cycle

A formal review process should be applied during all phases of life cycle with the collection of defect data and follow-up review to ensure corrections have been made.

Example Methods: Formal in-process reviews: defects (type and severity, time, source); defect removal efficiency; reliability predictions.

e. Conduct Comprehensive Tests

Testing is performed at all levels (unit, integration, system, alpha/beta) to ensure coverage of requirements. Test cases as determined from the test planning are executed, data collected on failures, continued testing determined through reliability estimation and programmatic cost/schedule constraints. Regression testing is conducted after any modifications due to development system testing or in-service support changes.

Example Methods: Test suites: unit, integration, system; Dynamic testing: functional, interface, performance, probabilistic, regression, stress, usability; equivalent classes to determine prioritization; reliability estimation from test data.

f. Use Failure Reporting and Corrective Action Procedures

Failure reporting and corrective action procedures should be used during development initiate the defect tracking process and to determine testing/retesting activities. This activity is continued during in-service support as part of the change management process.

Example Methods: FRACAS, reliability estimation.

g. Demonstrate Acceptance

Acceptance (qualification, certification, customer demonstration) testing/demonstration is conducted to ensure reliability case evidence is sufficient to meet customer and/or regulatory requirements.

Example Methods: Customer/certification authority system acceptance test suites; reliability case evidence review.

h. Sustain Reliability Throughout In-Service

Sustainment of an acceptable level of software reliability during in-service includes continued attention to reliability data collection and sustainment of the plan/case framework. Collect and analyze failure data during operational use to determine fault categories, sources of failure, and whether operational reliability claims are being met. Sustain/improve the reliability as software is modified during support. Ensure the support concept includes updates as necessary to the software reliability plan/case and associated reliability activities.

Example Methods: FRACAS, customer/certification authority system acceptance test suites; reliability case evidence review; repeat activities as necessary from the development phase.

4.3 Roles and Responsibilities

The roles and responsibilities for system/software reliability are illustrated in Figure 2. The Customer, Supplier, and Certification Authority have special relationships that will vary from project to project based on the application area, criticality of the system, and the formality of the engineering process.

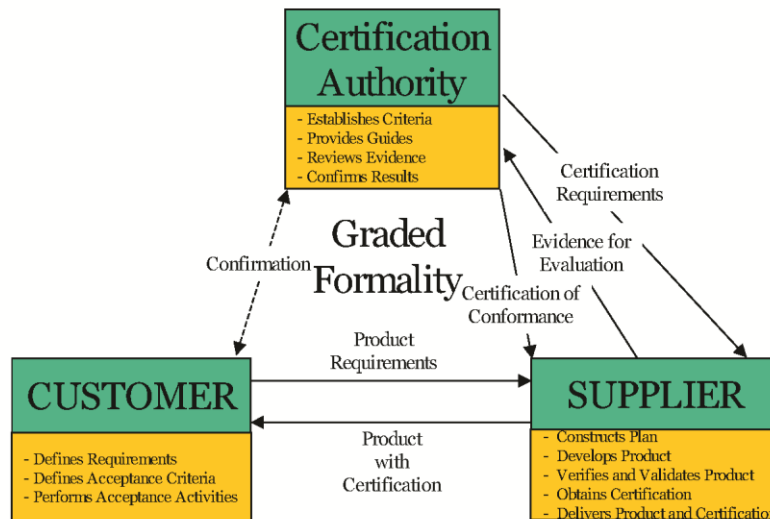


FIGURE 2—CUSTOMER-SUPPLIER-CERTIFICATION RELATIONSHIP

4.3.1 CUSTOMER

The customer represents an organization that procures a system, software product or software service from a supplier. The customer could be one or more of the following: buyer, owner, user, and/or purchaser. It is the responsibility of the customer to establish their requirements, work with the supplier to ensure their requirements are correctly interpreted, and provide appropriate review and approval of the reliability evidence that the delivered product satisfies the specified requirements.

In some cases where regulatory or certification authorities must approve use of the system, the customer will ensure the specified requirements include appropriate regulatory/certification concerns. In addition, the customer must ensure that required supplier interactions, demonstrations, and any other communications with such authorities is conducted as required. Confirmation of certification requirements is coordinated among the customer, supplier, and certification authority.

4.3.2 SUPPLIER/CONTRACTOR

The supplier represents an organization that enters into a contract with the customer for the supply of a system, software product or software service under the terms of the contract. The term "supplier" is synonymous with contractor, producer, seller, or vendor. The customer may designate a part of its organization as a supplier. The supplier may have many separate entities such as the contract developer, vendor that supplies OTS parts, and so forth.

4.3.3 CERTIFICATION/ACCEPTANCE AUTHORITY

The Certification Authority/Regulator is the organization or person responsible within the state or country concerned with the certification of compliance with the requirements. Certification is the legal recognition by the certification authority that a product, service, organization or person complies with the requirements. Such certification comprises the activity of technically checking the product, service, organization or person and the formal recognition of compliance with the applicable requirements by issue of a certificate, license, approval or other documents as required by national laws and procedures.

A software product may be certified as a separate item, a component part of a hardware component certification, or a software component part of a complete system depending on the application. The certification authority may use independent assessment personnel, or designees within the supplier organization that have been approved for the independent assessment role. Other combinations are possible depending on the country, customer organization, application, and critical nature of the software.

5. Task Activities

The purpose of this section is to outline specific task activities that will support achievement of a reliable software product within the context of the Software Reliability Program Standard [JA1002] and the system reliability standard and guidelines presented in references [JA1000] and JA1000-1]. Software reliability cannot be improvised. It must be addressed consistently and systematically, and as an integrated part of system reliability. Reliability must be addressed throughout the complete life cycle from concept through development, delivery and operational use, in a way that allows demonstration to the customer that reliability requirements are being met. Addressing reliability early in the life cycle can reduce the possibility of costly product redesign. The broad approach to ensure software reliability, is illustrated in Figure 3, and based on three simple principles:

- a. determine customer software reliability requirements;
- b. meet customer software reliability requirements; and
- c. demonstrate that the software reliability requirements have been satisfied.

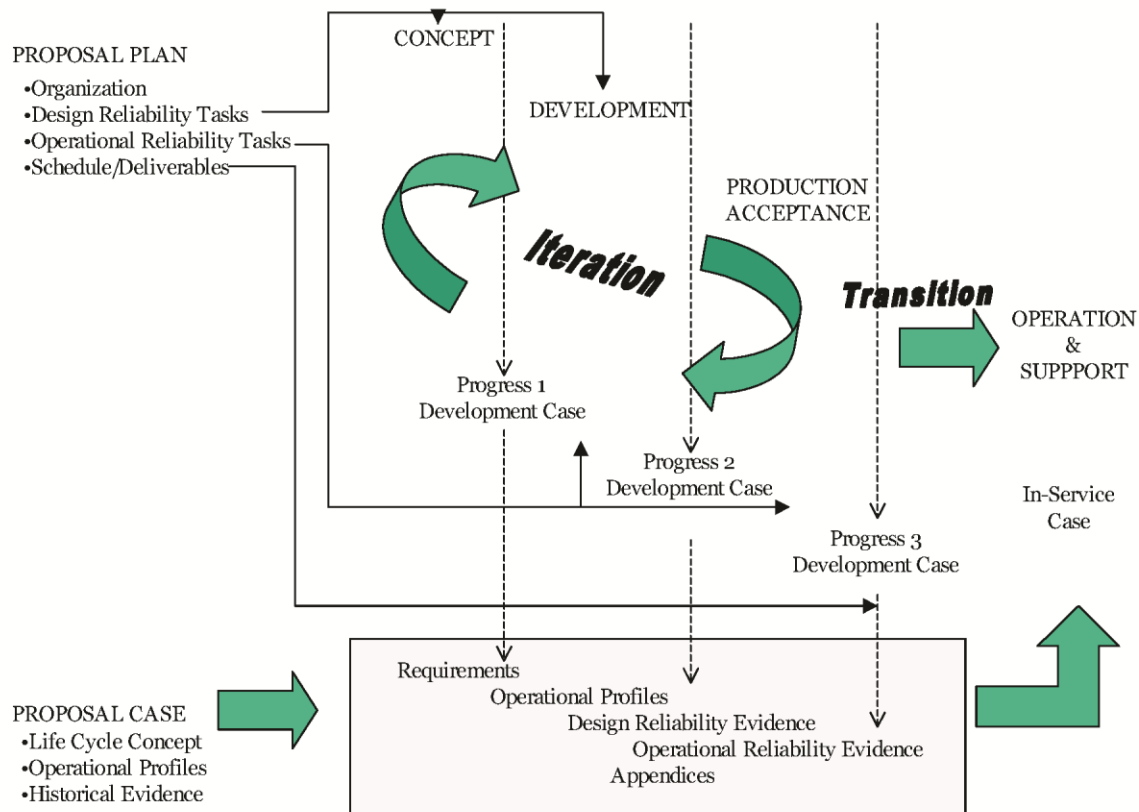


FIGURE 3—SOFTWARE RELIABILITY PROGRAM PLAN-CASE FRAMEWORK

A software reliability program defines the activities necessary to negotiate with the customer what the software reliability requirements are, identify and conduct analyses that ensure customer reliability requirements are met, and provide demonstrated evidence that the customer reliability requirements have been achieved. The identified activities along with measures of acceptance constitute the software reliability Plan. These activities include life cycle practices, methods, and processes that define how the software is designed to meet reliability as well as how the software is operationally tested to ensure that requisite reliability has been attained. The results gathered from the analysis activities constitute the software reliability Case.

A standard Plan and Case mechanism is the recommended approach to documenting a software reliability program [JA1002]. This Plan-Case structure should be implemented as early as possible in the life cycle and continued throughout the In-service phase. Suggested templates for the Plan and Case content are illustrated in Appendix B and described in more detail later in this section.

5.1 Reliability Analysis Tasks

Software reliability analysis tasks are defined within the context of achieving overall system reliability. The context for system reliability and the software reliability tasks presented in the following subsections is consistent with references [JA1000] and [JA1000-1]. The reliability analysis tasks are conducted at various points throughout the product life cycle phases depending on the iteration of the product design, changes to requirements, and the support concept.

A software reliability program should include management tasks such as: life cycle standard process standard (C.4.3) to define the overall software/quality engineering activities; requirements management (C.4.5) to elicit, analyze, document, and manage changes to the customer requirements and any derived requirements; risk management (C.4.6) to balance trade-offs among reliability cost, schedule, and performance; process assessment (C.4.4) to determine at milestone points whether processes, methods and techniques are effective and whether reliability goals appear to be achievable/achieved; configuration management (C.4.1) to control analysis results and reliability case evidence throughout the life cycle; and FRACAS (C.4.2) to ensure appropriate reliability data collection, analysis, and root cause can be conducted.

5.1.1 DETERMINE CUSTOMER REQUIREMENTS

Determining customer requirements requires customer-supplier negotiations, identification of the operational use profiles, definition of precisely what reliability means in terms of failures and criticality levels, and defining in-service conditions of use. Determining customer requirements analysis tasks will generally occur during the Concept and Feasibility phase with iteration across all other life cycle phases as necessary to update and manage changes to the requirements.

5.1.1.1 *Establish Customer-Supplier Dialogue*

Customer-supplier dialogue can be established through several activities that are a normal part of system requirements elicitation, analysis, and documentation. If certification and approval by a certification authority are also required, then the dialogue should also involve the certification authority. The key aspect of interest for this task is to determine the Goals for software reliability in the context of system reliability goals, what specific software reliability questions are to be answered (e.g., what does "failure" mean?), and precisely what type of evidence would be suitable for answering those questions.

Typical methods/techniques that might be used include: quality function deployment (C.1.11), goal-question-metric (C.1.6), formal scenario analysis (C.1.5), and various types of customer surveys and interactions intended to clarify information provided by the customer and/or the supplier.

5.1.1.2 *Identify Operational Conditions of Use*

One of the key activities of a reliability program is to determine the profiles of the product's operational use. For software this involves determining the complete set of operations (major system logical tasks) with their probabilities of occurrence. Typical methods/techniques that might be used include: formal scenario analysis (C.1.5), reliability bench marking (C.3.8) in combination with customer surveys and interactions with operational personnel.

5.1.1.3 *Define In-Service Conditions of Support*

It is important to understand the conditions under which the software is to be supported and how the support (e.g., field loading of software) might affect the approach for establishing a software reliability program. The references [JA1004], [JA1005], and [JA1006] provide a comprehensive approach to addressing software support and supportability. In addition, typical methods/techniques that might be used for software reliability aspects of support include: quality function deployment (C.1.11), formal scenario analysis (C.1.5), reliability bench marking (C.3.8), and various types of using/support organization surveys and interactions intended to clarify information provided by the customer and/or the supplier.

5.1.1.4 Establish Metrics: Goals, Assumptions and Claims, and Expected Evidence

Quantitative and qualitative measures of success need to be established as early as possible. These measures may vary over time as the requirements stabilize and the design is realized. Quantitative software reliability measures such as expected failure rate at delivery must be allocated from system reliability measures. Other quantitative software reliability measures such as defect removal efficiency or fault density must be derived from an understanding of the target failure rate as well as historical data. Qualitative measures need to be defined in terms of an iteration of demonstrated attention to lack of defects in the software specifications, architectural design, and code implementation. Evidence of quality management and engineering practices may be included in the qualitative measures. The negotiation and selection of these measures must be done very judiciously. Every measure must have a definite purpose in the decision-making processes otherwise the measures may become counterproductive. The evidence and associated measures will depend on the type of system. For example, possible failure measures of interest are illustrated in Table 5.

TABLE 5—EXAMPLE RELIABILITY FAILURE MEASURES BY APPLICATION TYPE

Metric	Example	Applicability
Failure On Demand	0.001 → 1 out of 1,000 service requests results in a failure	Transaction systems e.g., ATM or data communication switch
Rate of Failure Occurrence	0.002 → 2 failures are likely in 1,000 operational time units	Periodically operating systems e.g., CAD system, flight simulator, office automation software
Mean Time to Assist Rate of Assists	500 → the time between assists is 500 time units 0.002 → 2 assists are likely in 1000 operational time units	Production/manufacturing systems e.g., Microelectronics control software
Mean Time to Failure	1,800 → the time between failures is 1,800 time units	Continuously operating systems e.g., Nuclear reactor control system, life support system
Operational Availability	0.999 → the software is available for 999 out of 1,000 time units.	Continuously operating systems e.g., Nuclear reactor control system, life support system

Typical methods/techniques that might be used include: quality function deployment (C.1.11), goal-question-metric (C.1.6), formal scenario analysis (C.1.5), reliability allocation (C.1.12), reliability block diagrams (C.1.13), six sigma (C.1.16), SRE (C.1.20), statistical analysis (C.1.21), coverage analysis (C.3.3), dynamic test methods (C.3.4), formal in-process reviews (C.3.5), reliability estimation modeling (C.3.9), and process assessment (C.4.4).

5.1.1.5 *Develop Plan*

Once reliability requirements are reasonably understood through customer-supplier negotiation, the initial plan of reliability analysis tasks can be established. This plan may include tasks already completed or in progress, such as those tasks associated with determining customer requirements. The plan should be a documentation of the agreed-upon claims and evidence along with the selected analysis, design, and verification tasks that will be conducted. The plan should be approved by the customer and certification authority as appropriate. The Software Reliability Plan template illustrated in Appendix B provides an indication of information that might be included in the plan. Potential methods/techniques to support the task activities are outlined in Appendix C. These methods/techniques do not encompass all that exist. There are existing methods/techniques not mentioned in the appendix and new ones being developed that may be more suitable. However, the basic guidelines presented in this document should still provide a valid framework for selecting the most appropriate methods/techniques depending on the application and customer requirements.

5.1.1.6 *Document Pre-Development Case Evidence*

The collection of software reliability evidence created during the pre-development life cycle activities should be documented as the initial software reliability case. The Software Reliability Case template illustrated in Appendix B is an example format for information that might be included in the case. See 5.2 for more guidelines on the case documentation.

5.1.2 MEET CUSTOMER REQUIREMENTS

Meeting customer requirements requires definition of an appropriate software life cycle model that includes the planned reliability tasks, including interaction with systems engineering; conduct of static and dynamic analysis tasks with results reporting, performance of design, implementation, and test activities; continual assessment and management of reliability risk; and documentation of the development case of evidence. Meeting customer requirements tasks typically are the focus of the development and production life cycle phase, but may be used during the prototyping of design concepts as well as during the operation phase to show requirements are still being met and as part of the support activities to show updated software still satisfies the requirements.

5.1.2.1 *Define Life Cycle Model and Interaction with System Engineering*

One of the principles of this guideline document is that good software engineering practices embedded within a well-defined life cycle model will provide qualitative and quantitative evidence of improved software reliability. Qualitative evidence can be provided in the form of statistical correlation of specified practices with fewer delivered defects. Quantitative evidence can be provided in the form of reduced failures as observed in test results or in-service FRACAS (C.4.2) records.

The software engineering model should be an integrated part of the overall system engineering approach. System reliability activities will help define the scope of the software reliability program and its integration within the system/software engineering model. Management techniques of particular interest include: tailoring a life cycle model from appropriate life cycle standards (C.4.3); use of requirements management (C.4.5) and risk management (C.4.6) to manage an integrated approach to failure prevention, detection, removal, and mitigation; and process assessment (C.4.4) to identify system engineering areas of improvement. Other techniques that can be leveraged with system engineering management might include: common cause failure analysis (C.1.2), hazard analysis (C.1.7), probabilistic methods (C.1.9), probabilistic testing (C.3.4), reliability block diagrams (C.1.13), reliability prediction modeling (C.1.14), six sigma (C.1.16), software FMECA (C.1.18), software FTA (C.1.19), statistical analysis (C.1.21), reliability allocation (C.1.12), reliability bench marking (C.3.8), and reliability estimation modeling (C.3.9).

5.1.2.2 *Design Reliable Solution*

It is important to design software with the specific intent to be reliable. This requires considerable study and analysis of architectural options and translation of the options into a detailed design specification. The design of the selected architectural option should make it more efficient to demonstrate that requirements are satisfied or still satisfied if changes are made. Reliable architectural designs target characteristics of simplicity, modularity, and cohesion to facilitate the prevention, detection, removal, and mitigation of errors, faults, and failures of the software. Adopting an architecture and detailed design that includes interface verification methods (static and dynamic) facilitates defect prevention, identification, mitigation, and removal during both development and support activities. Software language selections to support implementation of such architectural design principles is an important consideration and should be done during design in order to enable a seamless transition to implementation, operation, and support. The language decision is a fundamental design decision that will affect production, testing, training, and support. Language implementation features to facilitate reliable solutions will be briefly discussed in 5.1.2.3.

Some of the techniques and methods that facilitate obtaining a reliable architectural design include: formal scenario analysis (C.1.5), reliability prediction modeling (C.1.14), response time, memory, constraint analysis (C.1.15), design by contract (C.2.1), engineering process maturity (C.4.4), fault tolerant design (C.2.2), information hiding (C.2.4), mistake/error proofing (C.2.5), reliability allocation (C.1.12), requirements analysis and traceability (C.4.5), boundary value analysis (C.3.1), cleanroom (C.3.2), peer reviews (C.3.7), and formal in-process reviews (C.3.5). When safety/security-critical requirements are important, techniques and methods that complement reliability assurance include: hazard analysis (C.1.7), Petri nets (C.2.6), software FMECA (C.1.18), software FTA (C.1.19), formal methods/languages (C.2.3), and I4 (C.2.4). Further discussion of safety and security considerations is in 6.2.

Nearly all of the Analysis and Design, and many of the Verification methods/techniques listed in Appendix C are possible selections for this part of a Software Reliability Plan. Many of the methods/techniques overlap or work in combination with other methods/techniques. It is important to tailor the selections to achieve a cost-effective reliability program.

5.1.2.3 *Implement Reliable Solution*

The process of implementing a design consists of translating the detailed design information into a software code form and then verifying that the implementation satisfies the design, and ultimately the specified requirements. To accomplish this it is necessary to maintain traceability of requirements (C.4.5) throughout the entire software life cycle process. Typical life cycle activities of coding, unit testing, analyzing various coverage measures, and conducting reviews should include attention to reliability-specific methods and techniques. Operational and support procedures should be developed to ensure software is reliably loaded, initialized, initiated and, as appropriate, replaced during field use.

The implementation may vary depending on the target language, operational environment, and executable medium. Reference [XTALK03] contains several articles that describe languages features, language selection, and implementation principles that may assist or hinder a reliable, safe, and secure solution. In particular, for most of the techniques discussed in Appendix C the language selection can facilitate or hinder the use of the technique. In some cases, the language completely or partially implements the technique. Language features such as strong type checking, assertion checks, formal exception handling, modularity constructs, compiler validation, and multiple platform implementations facilitate and/or implement such design techniques as: design by contract (C.2.1), degraded mode/fault tolerant capabilities (C.2.2), formal methods/languages (C.2.3), I4 (C.2.4), mistake proofing (C.2.5), requirements analysis and traceability (C.4.5), and nearly all of the verification techniques (C.3) appropriate for the lower level verification activities. In addition, these implementation language features and a support system of robust automated tools can greatly aid key reliability analysis techniques such as Petri nets (C.2.6), probabilistic methods and risk assessment (C.1.9 and C.4.6), response time/memory/constraint analysis (C.1.15), software FMECA (C.1.18), software FTA (C.1.19), software reliability engineering (C.1.20) and statistical analysis (C.1.21).

If the software is implemented in hardware, such as Application Specific Integrated Circuits (ASICs), then hardware (e.g., hardware description language with commercial tools for model checking and verification of designs, fabrication technology, built-in tests, production and acceptance tests) techniques must be integrated as well to provide evidence of meeting reliable functional, electrical, and environmental requirements.

The bottom line is to implement a software solution that is verified at the lowest levels against the design intent. It is also important that the implementation provides features to assist the analysis, verification, validation, and continued evaluation that demonstrates the software meets its customers' requirements.

5.1.2.4 *Verify and Validate Solution is Reliable*

Once the software is implemented and lower level verification has been achieved, it is important to verify and validate that the software solution interfaces with a representative operational system environment as required and expected. This involves integration and system testing, use of operational profiles, coverage analysis within the scope of the operational profiles, and specific reviews to ensure any special cases (such as non-reproducible hardware error processing or safety/security functions) are adequately verified. Validation of execution results against experimental data or known historical and/or operational data is an important activity. Software reliability prediction and estimation provide key decision information and facilitate identification of risks and appropriate risk mitigation strategies. All the verification techniques (C.3) are candidates for use, but those of particular importance include: boundary value analysis (C.3.1), coverage analysis (C.3.3), multiple testing techniques (C.3.4), software reliability estimation modeling (C.3.9), and software reliability prediction modeling (C.1.14). These verification techniques provide data for analyses critical to understanding the reliability of the software component as well as the system.

5.1.2.5 *Assess and Manage Reliability Risk*

The risk management (C.4.6) approach should include methods to assess how well reliability requirements are being achieved throughout the development (and support) phases. Typical methods/techniques such as system/software FMECA (C.1.18) and system/software FTA (C.1.19) establish potential high risk areas. Methods/techniques such as reliability prediction modeling (C.1.14), reliability estimation modeling (C.3.9), and the numerous static and dynamic testing methods provide inputs to the risk assessment. Methods/techniques such as probabilistic methods (C.1.9), probability/reliability-based risk assessment (C.4.6), and statistical analysis (C.1.21) provide confidence measures that support risk assessments. More comprehensive software reliability program techniques such as SRE (C.1.20) in conjunction with the use of formal scenario analysis/operational profiles C.1.5) provide a fundamental process within which most of the above techniques can be integrated to assess and manage reliability risk.

5.1.2.6 *Document Development Case Evidence*

The collection of software reliability evidence created during the development life cycle activities should be documented and appended to the initial software reliability case defined in 5.1.1.6. The Software Reliability Case template illustrated in Appendix B is an example format for information that might be included in the case. See 5.2 for more guidelines on the case documentation.

5.1.3 DEMONSTRATE CUSTOMER REQUIREMENTS

Demonstrating that customer requirements have been met involves the more formal aspects of qualification of the product and/or processes for customer and/or certification acceptance; establishing process controls to ensure reliability requirements can be sustained; transitioning the software product to the operational environment; training end-users and support staff; ensuring the capability to pursue continuous improvement is established; establishing a sustaining data collection capability to track failures and other pertinent reliability information; and documenting the post-development software reliability case for ultimate delivery to the customer support organization. Demonstrating customer requirements are satisfied is a focus of post-production and delivery for In-Service support, but selected demonstration activities may occur throughout the life cycle.

5.1.3.1 Qualify/Certify the Product and Process

The software reliability case should provide the required evidence that the software product, and the life cycle processes used to produce the product, has met all customer reliability requirements. The complete set of verification and validation evidence, of which the software reliability case is potentially a critical subset, is required for complete product qualification and certification per the certification authority requirements. Evidence and perhaps specific demonstrations that the software product satisfies the customer operational scenarios/profiles within a representative system operational environment is a critical part of the qualification and certification activities.

5.1.3.2 Establish Process Controls

The qualified software product must be controlled through a formal configuration management system (C.4.1). This system includes: (1) complete identification of the software product (including life cycle artifacts such as requirements, design, implementation, test, build); (2) a change management capability for controlling failure information, change requests, and traceability for software product versions; and (3) a transition and migration process for controlling distribution of the software product from the supplier to the customer. All of this would be part of a more comprehensive software support concept in which the software reliability plan and case construct would be sustained as the software product is modified.

5.1.3.3 Transition to Operational Environment

The actual transition of the software product from the supplier to the customer's operational environment (initial version or subsequent updated versions) should be done in accordance with a well-defined migration plan. This plan would include the transition of the software reliability plan/case information to the software support agency as well as training on what is required to sustain the software reliability program during field installation, operational testing and use, and logistic functions that are part of the software support concept. Support tasks may be a root cause for failures that should be considered as part of an SFMEA (C.1.18) or SFTA (C.1.19). Support tasks may require their own Formal Scenario Analysis (C.1.5) to identify potential activities that may have a higher risk of affecting the operational use of the software, such as the field loading of the software. See references [JA1004], [JA1005], and [JA1006] for more information on software supportability and support scenarios.

5.1.3.4 Train End-Users, Operations and Support Staff

The human operators, users, and support staff are an important component of a reliable system. These personnel will generally require training on the proper use of the system, including any interfaces with the associated software product(s). The software reliability program should include activities to identify any training requirements, analyze the most effective presentation method, develop appropriate training materials, and as appropriate, train those personnel who might affect the correct operation of the software. This training program should be developed as part of the customer requirements negotiation, implemented concurrently with the software product, improved through prototype interactions with the customer, demonstrated to the customer through official acceptance training activities, and updated/conducted as necessary by the support agency.

5.1.3.5 *Pursue Continuous Improvement*

The techniques of customer feedback/surveys for lessons learned, quality metrics (C.1.6 and C.1.11), common cause analysis (C.1.2), reliability estimation (C.3.9), reliability prediction (C.1.14), reliability-based risk assessment (C.4.6), and many others provide valuable feedback for root cause analysis (C.3.10) and general continuous improvement. Comprehensive management techniques such as process assessment (C.4.4) and engineering process maturity provide well-defined methods for pursuing continuous improvement. Software reliability case evidence is a primary source for establishing baseline measures and improvement objectives. The software reliability program is well-suited to tracking improvement goals and whether they have been achieved.

5.1.3.6 *Establish Data Collection and Reporting*

Formal data collection and reporting should be part of a systems approach typically characterized as a Failure Reporting and Corrective Action System (FRACAS), see C.4.2. Suggested data reporting concepts and existing data repositories are briefly discussed as a special consideration in Section 0.

5.1.3.7 *Document Post-Development Case Evidence for In-Service Use*

The collection of software reliability evidence created during the post-development life cycle activities related to customer acceptance and demonstrated capability should be documented as the post-development software reliability case. This case information should be appended to the software reliability case defined in 5.1.2.6 as the initial In-Service software reliability case and delivered to the customer support organization for sustained use. The Software Reliability Case template illustrated in Appendix B is an example format for information that might be included in the case. See 5.2 for more guidelines on the case documentation.

5.2 **Reliability Case Documentation**

The software reliability case captures the assumptions, claims, arguments, and evidence necessary to provide a convincing argument that a software product has a specified reliability. This collection of information, referred to as the Software Reliability Case, must be accurate, current, and complete and presented in a convincing manner in order to obtain sign-off by the customer and/or certifying authority. It consists of six components: system context description; software reliability goals, objectives, and requirements; assumptions and claims; evidence and reasoned arguments to support the claims; conclusions / recommendations; and certification records. There are essentially two types of evidence: (1) direct evaluation of the achieved reliability of the software products, and (2) evidence of the general suitability of the software engineering process. See Appendix B for a sample software reliability case file outline and FRACAS (C.4.2) for a discussion of case interface with a data collection and storage capability. This section provides an overview of the software reliability case as illustrated in Figure 4. Additional suggestions for reliability case information specific to life cycle phases is contained in reference [DEFS0042-3].

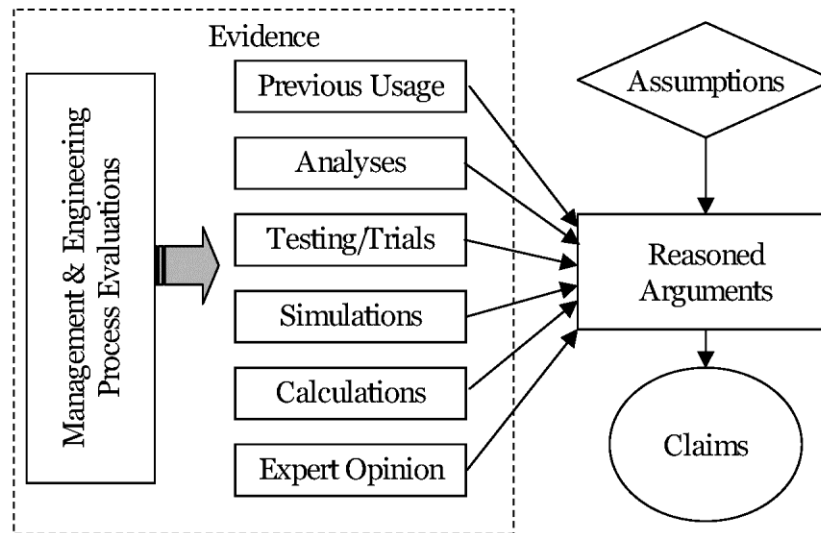


FIGURE 4—RELIABILITY CASE: CLAIMS BASED ON EVIDENCE

5.2.1 SYSTEM CONTEXT DESCRIPTION

The system description should provide a context for understanding the software component(s) in relation to the system. The description should briefly include the: system's physical equipment characteristics; system's physical boundary such as through block diagrams; system's primary role or function and any secondary roles in order to provide a view of its operational criticality – e.g., typical peacetime and wartime mission profiles; system's operating environments; interfaces with any equipment associated with the inputs, outputs, and services the software provides to the system; configuration build of the system/software for which the case evidence applies; personnel skill levels and training that are required to operate and/or maintain the equipment/software; and the maintenance policy/support concepts for each of system's role or mission profiles.

5.2.2 GOALS, OBJECTIVES, AND REQUIREMENTS

The information in the software reliability case file must correlate with the specified software reliability requirements. Hence, the software reliability goals, objectives, and specific requirements should be stated first, for the system and individual partitions, as appropriate. The process by which the reliability requirements were derived and apportioned to software should be described. The relationship between the system and software reliability requirements should be explained. Any regulatory and/or contractual reliability requirements should be highlighted. In addition, the agreed upon validation and certification criteria should be noted. Any risk areas associated with the system/software satisfying the reliability requirements should be identified along with how these risks will be, or have been managed. Based on the risks identified, the maturity of the proposed design solution and the reliability requirements, the strategy adopted to meet the requirements and provide the necessary assurance should be summarized. This strategy is justification for the reliability program activities and the identification of the success criteria for these activities.

5.2.3 ASSUMPTIONS AND CLAIMS

All assumptions, such as citing existing systems or research, and claims made relative to achievement and assessment of the software reliability goals and objectives should be clearly stated and justified. A good record in the past is not sufficient as evidence of good performance to be expected in an environment that is modified even slightly. Any change in the physical, logical, or data environment must be taken into account.

5.2.4 EVIDENCE

Three categories of evidence should be collected iteratively throughout the life cycle and included/referenced in the software reliability case file: process activities, product characteristics, and qualifications of people and resources that demonstrate achievement of software reliability requirements. This information can be summarized in terms of activities conducted, techniques used during the activities, and measures to satisfy claims across process, product, and resources and reliability control areas as illustrated in Table 6:

- a. Fault Elimination—Requirements, design, implementation and test prevention/detection/removal/mitigation of defects/faults;
- b. Failure Containment—Failure impact limitation; failure isolation and recovery; fault tolerant, degraded mode operations; operational profile coverage testing & analysis; and
- c. Failure Rate Estimation—Failure rate prediction from development parameters; FRACAS data collection; failure data analysis & reliability estimation modeling; statistical analysis and probabilistic methods; failure/impact reliability risk assessment

Adequacy of evidence provided for each activity should be evaluated. Professional judgment is required to evaluate the evidence presented. A checklist adapted from [DEFS0042-3] is indicated below as a guide to some of the possible evaluation areas:

- a. stated objectives for the activity are clearly defined;
- b. activity is systematic and complete;
- c. activity been undertaken at a time that will allow influence on the design;
- d. usage profile and environmental constraints have been considered for the activity;
- e. physical and functional boundaries of the activity have been defined;
- f. any assumptions are defined (e.g., inputs from other systems or services), and are realistic and reasonable;
- g. justification is provided for the activity method/technique used and is reasonable;
- h. description is provided of who (e.g., user, maintainer, designer etc) was consulted during the activity and these personnel and level of consultation is reasonable;
- i. activity recommendations are clearly defined and reasonable;

- j. documentary evidence indicates that the recommendations have been implemented; and
- k. activity results have been progressively updated to reflect the latest design and are being used as an input to design reviews.

TABLE 6—EXAMPLE FORMAT FOR SUMMARY OF SOFTWARE RELIABILITY CASE EVIDENCE

System: _____

Intended Use/Environment: _____

Phase/Date: _____

Reliability Control Measure	Product Evidence/ Safeguards	Process Evidence/ Safeguards	Resource Evidence/ Safeguards
Fault Elimination	- - - -	- - -	- - -
Failure containment	- - - -	- - -	- - -
Failure rate estimation	- - - -	- - -	- - -

5.2.4.1 Process Activities that Demonstrate Achievement of Software Reliability Goals

A description of the selected life cycle model and development method should be provided, including an explanation of how this model and method contribute(d) to the attainment and assessment of reliability goals throughout the life cycle phases. Specific life cycle activities that were used to assess software reliability should be called out, such as performing iterative risk analyses or using static analysis techniques. An assessment should be made of the:

- software reliability design analysis;
- software reliability code analysis;
- software reliability change analysis; and
- effectiveness of validation and verification activities.

Suspected or confirmed reliability problems should be documented, along with the current status of their resolution. Results from analyzing and interpreting process metrics should also be discussed.

5.2.4.2 *Product Characteristics that Demonstrate Achievement of Software Reliability Goals*

A description of the design features that contribute to enhanced reliability should be provided, such as: design by contract (C.2.1), I4 (C.2.4), and system/software fault tolerance design (C.2.2). This description should explain how the likelihood of common cause failures (C.1.2) has been eliminated or reduced, through use of techniques such as system/software FMECA (C.1.18) and system/software FTA (C.1.19). In addition, a discussion of whether the product: (1) operates in a demand-mode or continuous-mode environment; (2) was designed to fail safe or fail operational; and (3) contains any monitoring and/or error detection and correction features should be included. The results of static and dynamic test analyses should be recorded, along with an analysis of the effectiveness of the reliability control measures. Results from analyzing and interpreting product metrics should also be discussed. A well-planned dynamic test program is an essential part of providing confidence in the reliability of software. Dynamic testing evidence (C.3.4) should be provided along with coverage analysis (C.3.3) and evidence that requirements have been traced (C.4.5) to design, implementation, and dynamic tests.

5.2.4.3 *Qualifications of People and Resources that Demonstrate Achievement of Software Reliability Goals*

An explanation of why the education, experience, and certification of the professional staff are appropriate for a project of this reliability level should be provided. Likewise, a justification of why the hardware and software platforms, including automated tools, are appropriate for this project should be provided. Results from analyzing and interpreting people/resource metrics should also be discussed.

5.2.5 CONCLUSIONS/RECOMMENDATIONS

The conclusions/recommendations should summarize the key information presented in case evidence accumulated to date, such as which software reliability requirements have/have not been met and with what level of confidence. In interim issues, it should recommend whether the project should proceed to its next milestone, or what further work is required to enable the project to progress. In addition, it should recommend what activities should be conducted in the future in order to generate the necessary assurance that the reliability requirements will be satisfied and what residual risks/limitations may constrain the results. For operational use, recommendations should be provided regarding the software qualification/certification status.

5.2.6 CERTIFICATION RECORDS

An accurate and complete chronological history of all certification-specific information and activities should be included or referenced in the software reliability case file. The format and precise content is typically dependent on the certification authority. An example of the type of Federal Aviation Administration (FAA) certification authority interactions and a fragment of potential certification information is provided in Appendix E.

6. Special Considerations

A software reliability program may involve a variety of special considerations, some of which are discussed in this section.

6.1 Tailoring the Software Reliability Program

A layered use of standards and guidelines with appropriate tailoring of activities and techniques is the most effective approach to establishing a software reliability program. Within each of the layers, such as illustrated in Figure 5, there are opportunities to tailor the activities to best suit the application. This tailored set of activities is negotiated as part of the initial negotiation with the customer to determine system/software reliability requirements.

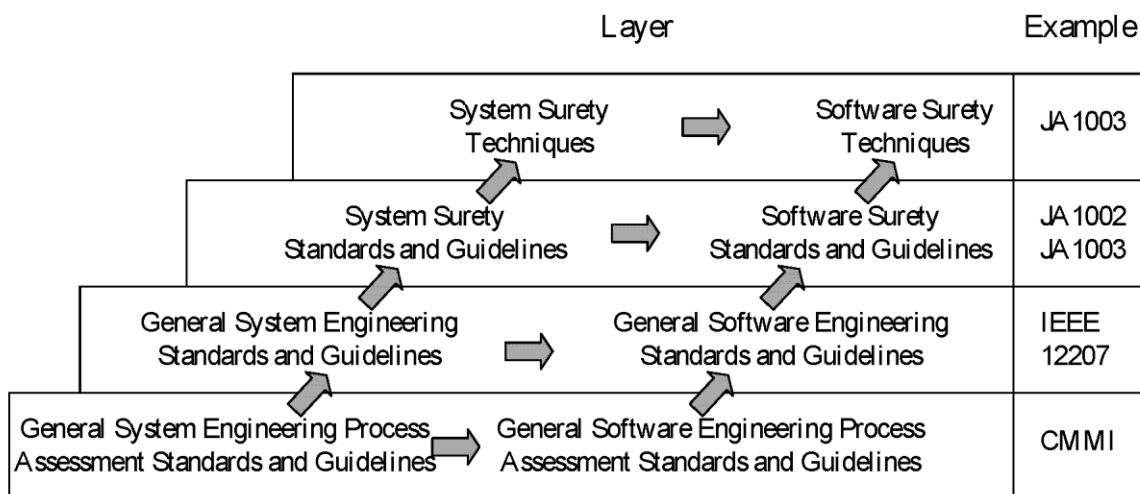


FIGURE 5—LAYERED APPROACH TO SOFTWARE RELIABILITY TAILORING

The tailoring of a software reliability program for a given project will be the focus of implementing the "Determine Customer Requirements" reliability analysis task. The examples in Appendix D and Appendix E provide some instances of tailoring the analysis tasks. Some specific tailoring considerations are discussed in the following subsections. Special tailoring considerations for application variations and complexity, criticality levels, technique selection, safety and security, Off-The-Shelf software, and data collection and reporting are briefly discussed in 6.2, 6.3, and 6.4.

6.1.1 APPLICATION VARIATIONS AND COMPLEXITY

There are many different applications for which the drivers for tailoring will vary. Such applications include military, aerospace, motor vehicle, railway, medical equipment, pharmaceutical, and nuclear industry. Regulations (requirements specific to the domain) are typically developed and managed by an application domain organization. The application regulations will drive the specific system/software engineering processes, verification and validation activities, and in general the reliability, safety, and security techniques that provide the required assurance. Within the context of the application domain regulations, the required integrity level and complexity of the system/software will determine which reliability techniques will be most appropriate to provide the required level of assurance.

Military systems will be driven by the military acquisition system procurement regulations, responsible armed forces regulations, and the subsequent supplier contractual mechanism. Available funding will be driven by government budget reviews and military allocations. The United States Department of Defense military acquisition process also varies from other country's (e.g., United Kingdom Ministry of Defence, Canadian National Defense) military acquisition. Some military/federal regulations will apply as well as industry-specific standards and guidelines. The acquisition system procurement process must require reliability/safety/security plan and case information to be generated as a Key Performance Parameter if it is to be assured. In addition, the requirement should be for acquisition customer-supplier negotiation, analysis, and demonstrated assurance evidence throughout the acquisition and support life cycle. For system acquisition this is directly supported by the system reliability program standard/guideline, [JA1000] and [JA1000-1]. For the software acquisition this is directly supported by the software reliability program standard/guideline, [JA1002] and this JA1003 guide.

Industry systems will be driven by the full range of customer priorities and the enterprise market requirements for time-to-market and profit. An additional driver is the potential for litigation if the system/software product reliability/warranty does not meet expectations. Industry applications such as aerospace, railway, medical, and nuclear energy have significant safety and security concerns. Legal liability can fall on the manufacturer, supplier, distributors, or certifier of products. Such systems/software may have a certification authority provide a required oversight function:

1. Federal Aviation Administration for commercial airlines;
2. Nuclear Regulatory Commission for the nuclear power industry;
3. Her Majesty's Railway Inspectorate for the United Kingdom railway systems; and
4. Food and Drug Administration for pharmaceutical equipment and processes.

Each of these certification authorities will have source regulations, evaluation and assessment procedures, and best practice recommendations that will need to be integrated into the system/software reliability plan/case framework. Associated software will need to apply techniques such as hazard analysis (C.1.7), software FMECA (C.1.18), software FTA (C.1.19), I4 principles (C.2.4), and perhaps even formal methods and languages (C.2.3) to provide adequate assurance that the systems are safe and reliable. When a system/software has significant reliability, safety, or security concerns independent evaluations, assessments, and audits may be part of the required oversight activities. A partial example of establishing a software reliability program for an aerospace application within the requirements of the Federal Aviation Administration is illustrated in Appendix E.

A system/software product may be very complex. This complexity can originate in the nature of the problem being solved by the application, number of functions being provided, number of components with which interfaces are required, or number of applications for which the product is intended to be used. In turn, this complexity manifests itself in the number of system/software requirements, architecture design structure, implemented product decision paths, and level of static and dynamic testing required. A well-planned and executed software reliability program should be tailored to reduce these complexities.

6.1.2 CRITICALITY LEVELS

There are many different concepts for criticality levels related to system/software reliability. The term itself may be "severity", "integrity", "risk", or just "criticality" level. In almost all cases the criticality level is a qualitative ordered scale. The usual association with the criticality level is that the rigor of the reliability evidence required increases with the criticality levels. Specific system/software engineering and reliability/safety-specific activities are defined so there is acceptable assurance the desired criticality level will be attained. Although there are standards and guidelines (e.g., [DO178B]) that specify which activities are to be conducted to satisfy the desired criticality level, such activities are still part of a tailoring process involving the customer and supplier, and as appropriate the certification authority. Thus, the specific task to "Determine Customer Requirements" would involve the specific definition of what failures are of concern, the criticality levels associated with those failures, and what specific reliability tasks must be conducted to provide the required confidence that such failures will not occur.

For example, criticality levels could be based on: level of financial cost impact to the customer, level of functional impact to the customer; reliability/failure rate level; or risk level as a qualitative combination of frequency of failure and impact of failure for each failure category. What is certain is that additional reliability concerns of safety and/or security will increase the level of rigor required by the reliability program. The customer/supplier negotiations are critical to establishing the desired criticality levels. A software reliability criticality level matrix with example software applications is illustrated in Table 7.

TABLE 7—EXAMPLE SOFTWARE RELIABILITY CRITICALITY MATRIX

↑	Catastrophic				Nuclear Weapon/ Safety of Flight
	Hazardous/Severe			Nuclear Energy	
	Major		Medical Radiation Treatment		
	Minor	Information Management			
Hazard Level		10^{-1}	10^{-3}	10^{-5}	10^{-6}
Failure Rate		→			

Table 7 is only meant to be illustrative since within the indicated applications, the desired failure rate and hazard level combination (risk level) may be different than indicated.

At the “upper” levels of criticality there is the problem of providing sufficient quantitative evidence of very high (e.g., less than 10^{-6} failure rate) or even ultra high (e.g., less than 10^{-9} failure rate) reliability systems. For a given confidence interval, the uncertainty in the reliability estimate determined from testing depends on the number of samples gathered. The more samples, the smaller the uncertainty in the estimate. As an example, for a 95% confidence interval and 20 samples, the reliability estimate would range approximately from 0.70 to 1.60 of the measured average value. However, for 200 samples, the range would be approximately 0.90 to 1.15 of the average.

If the reliability requirement is high, say, no more than 1 failure in 100,000 hours of operation, it could take more than 11.4 years of continuous testing to gather a single data point. A minimum of 20 samples is required to achieve the 95% confidence degree of certainty. Hence somewhere in the order of 228 years of testing would be required. However, if large scale parallel independent testing on multiple units is possible, or the software execution hours are significantly less than the system operational hours, then it may be feasible to demonstrate high reliability through testing. In addition, it may be that only a small portion of the software is required to have high reliability, and can be so demonstrated using these accelerated testing techniques. See the Figure 6 below for an illustration of determining the relationship among defect sample size, reliability multiplier, and confidence limits.

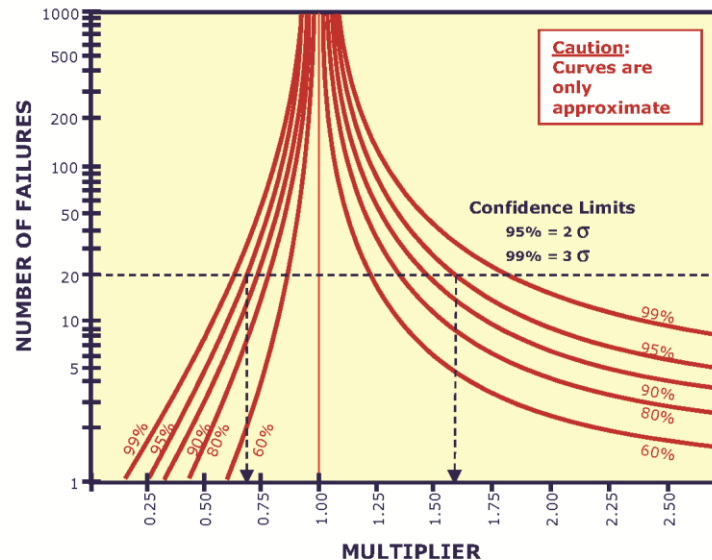


FIGURE 6—RELIABILITY CONFIDENCE LIMITS

6.1.3 TECHNIQUE SELECTION

One of the problems that will need to be resolved by an effective software reliability program is which techniques to select from the many that exist. There are many techniques illustrated in Appendix C, and other existing techniques, and more techniques that will exist in the future. Selecting which techniques will be best suited for a specific software reliability program will be a key activity of the tailoring approach.

There are many decision-theory selection methods to assist in determining which technique(s) from a sample of alternative techniques is more preferable to use. One example is the Pugh selection method (C.1.10). As an example, such a method might be used in combination with selection criteria such as defined in [AIAAR013] or [IEEE1413] for evaluation of alternative software reliability models. In [AIAAR013] the following model evaluation criteria are suggested:

1. model predictive validity for accuracy, trend, bias, and noise;
2. ease of measuring model parameters, including cost and schedule impact;
3. quality of assumptions in comparison with the real world as well as adaptability to special conditions and environments;
4. capability to estimate useful quantities needed by software project personnel;
5. applicability to program evolution and changes in the test and operational environment;
6. simplicity of understanding the concept, data collection, program implementation, and validation; and
7. insensitivity to small differences in input data and parameters without losing responsiveness to significant differences.