IEEE Std 1219-1993

IEEE Standard for Software Maintenance

Sponsor

Software Engineering Standards Subcommittee

of the

IEEE Computer Society

Approved December 3, 1992

IEEE Standards Board

Abstract: The process for managing and executing software maintenance activities is described.
Keywords: maintenance, software, software maintenance

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

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

PDF: ISBN 0-7381-0418-3, SS15867

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

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

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

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

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

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

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

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

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

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Introduction

(This introduction is not a part of IEEE Std 1219-1992, IEEE Standard for Software Maintenance.)

This standard describes the process for managing and executing software maintenance activities. Clause 1 lists references to other standards useful in applying this standard. Clause 2 provides a set of definitions and acronyms that are either not found in other standards, or have been modified for use with this standard. Clause 3 contains required information pertaining to the software maintenance process. In order to be in compliance with this standard, clause 3 must be adhered to.

Annex A, Maintenance guidelines, contains additional information that is not required for compliance. Topics in this annex include: the source of maintenance forms discussed in the standard, validation and verification (V&V), software quality assurance, risk assessment, safety, security, software configuration management (SCM), metrics, software replacement policy, and the maintenance process. Annex B, Supporting maintenance technology, includes the topics re-engineering, reverse engineering, reuse, maintenance planning, impact analysis, and software tools.

The audience for which this standard is intended consists of software development managers, maintainers, software quality assurance personnel, SCM personnel, programmers, and researchers.

This standard was developed by a working group consisting of the following members who attended two or more meetings, provided text, or submitted comments on more than two drafts of the standards.

D. Vera Edelstein, *Chair*
Salvatore Mamone, *Secretary*

Shawn A. Bohner
Rita Costello
Robert Dufresne
Frank Douglas
Gregory Haines
Alan Huguley
Moisey Lerner
Julio Gonzalez Sanz
Ben Scheff
Carl Seddio
Ahlam Shalhout
Malcolm Slovin
Edwin Summers
Richard Weiss
Ralph Wootton

The following individuals also contributed to the development of the standard by attending one meeting or providing comments on one or two drafts.

Ray Abraham
Joseph Barrett
James E. Cardow
Neva Carlson
Sharon Cobb
Sandra Falcone
Mari Georges
Meinhard Hoffman
T. S. Katsoulakos
Thomas Kurihura
Robert Leif
Suzanne Leif
K. C. Leung
Yuan Liu
John Lord
Kenneth Mauer
Basil Sherlund
Harry Sneed
Harry Trivedi
Bob Walsh

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

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

Ron Berlack	Robert Kosinski	Hans Schaefer
William Boll	Thomas Kurihara	David Schultz
Fletcher Buckley	Robert Lane	Gregory Schumacher
Kay Bydalek	Boniface Lau	D.M. Siefert
Evelyn Dow	Moisey Lerner	Malcom Slovin
Einar Dragstedt	Ben Livson	Al Sorkowitz
Vera Edelstein	Joseph Maayan	Vijaya Srivastava
Caroline Evans	Kukka Marijarvi	Richard Thayer
John Fendrich	Roger Martin	George Tice
Kirby Fortenberry	Ivano Mazza	Leonard Tripp
Yar Gershkovitch	James Perry	Dolores Wallace
David Gustafson	Andrew Sage	William Walsh
John Harauz	Julio Gonzalez Sanz	Robert Wells
John Horch	Stephen Schach	Natalie Yopconka
Roy Ko		Janusz Zalewski

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

Marco W. Migliaro, *Chair* **Donald C. Loughry,** *Vice Chair*
Andrew G. Salem, *Secretary*

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

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

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

Christopher J. Booth
IEEE Standards Project Editor

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Contents

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

IEEE Standard for Software Maintenance

1. Overview

1.1 Scope

This standard describes an iterative process for managing and executing software maintenance activities. Use of this standard is not restricted by size, complexity, criticality, or application of the software product. This standard uses the process model, depicted in table 2, to discuss and depict each phase of software maintenance. The criteria established apply to both the planning of maintenance for software while under development, as well as the planning and execution of software maintenance activities for existing software products. Ideally, maintenance planning should begin during the stage of planning for software development (see annex clause A.3 for guidance).

This standard prescribes requirements for process, control, and management of the planning, execution, and documentation of software maintenance activities. In totality, the requirements constitute a minimal set of criteria that are necessary and sufficient conditions for conformance to this standard. Users of this standard may incorporate other items by reference or as text to meet their specific needs.

The basic process model includes input, process, output, and control for software maintenance. Metrics/ measures captured for maintenance should enable the manager to manage the process and the implementor to implement the process (see table 3). This standard does not presuppose the use of any particular development model (e.g., waterfall, spiral, etc.).

This standard provides additional software maintenance guidance on associated topics in annex A and tools/ technology assistance in annex B.

1.2 References

The following standards are directly referenced in this standard. Table 1 provides a cross-reference of IEEE standards that address various topics related to software maintenance. These standards are binding to the extent specified within this standard and are referenced to avoid duplication of requirements.

IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology (ANSI).[1]

IEEE Std 730-1989, IEEE Standard for Software Quality Assurance Plans (ANSI).

IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans (ANSI).

IEEE Std 829-1983 (Reaff 1991), IEEE Standard for Software Test Documentation (ANSI).

IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software (ANSI).

IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software (ANSI).

IEEE Std 983-1986 (w1992), IEEE Guide for Software Quality Assurance Planning (ANSI).[2]

[1]IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.
[2]IEEE Std 983-1986 has been withdrawn; however, copies can be obtained from the IEEE Standards Department, IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans (ANSI).

IEEE Std 1028-1988, IEEE Standard for Software Reviews and Audits (ANSI).

IEEE Std 1042-1987, IEEE Guide to Software Configuration Management (ANSI).

IEEE Std 1058.1-1987, IEEE Standard for Software Project Management Plans (ANSI).

IEEE Std 1074-1991, IEEE Standard for Developing Software Life Cycle Processes (ANSI).

1.3 Terminology

The words *shall* and *must* identify the mandatory (essential) material within this standard. The words *should* and *may* identify optional (conditional) material. The terminology in this standard is based on IEEE Std 610.12-1990.[3] New terms and modified definitions as applied in this standard are included in clause 2.

Table 1—The relationship of IEEE software engineering standards to IEEE Std 1219-1992

Relationship		**IEEE standard**
Process	Problem ID/classification	
	Analysis	830-1984, 1074-1991
	Design	830-1984, 1016-1987, 1074-1991
	Implementation	1008-1987, 1074-1991
	System testing	829-1983, 1074-1991, 1028-1988, 1012-1986
	Acceptance testing	1074-1991, 1012-1986
	Delivery	
Control	Problem ID/classification	
	Analysis	830-1984
	Design	830-1984, 1016-1987
	Implementation	829-1983, 1008-1987
	System testing	829-1983, 1028-1988, 1012-1986
	Acceptance testing	829-1983, 1028-1988, 1012-1986
	Delivery	1063-1987
Management	Configuration management	828-1990, 1042-1987
	Measurement/metrics	982.1-1988, 982.2-1988
	Planning	829-1983, 1012-1986, 1058-1987
	Tools/techniques	
	QA	730-1989, 983-1986 (w)1992)
	Risk assessment	730-1989, 982.2-1988
	Safety	
	Security	

[3]Information on references can be found in 1.2.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

1.4 Conventions

The conventions used in each figure depicting a maintenance phase are shown in figure 1.

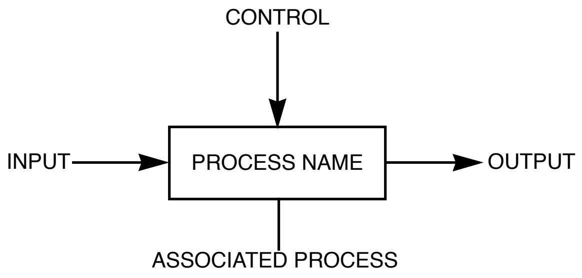

Figure 1—Conventions

The term *associated processes* refers to external processes that are defined in other standards; i.e., software quality assurance, software configuration management (SCM), and verification and validation (V&V). The term *associated processes* also refers to the metrics process illustrated within this document.

2. Definitions and acronyms

2.1 Definitions

The definitions listed below establish meaning in the context of this standard. These are contextual definitions serving to augment the understanding of software maintenance activities as described within this standard. Other definitions can be found in IEEE Std 610.12-1990.

2.1.1 adaptive maintenance: Modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment.

2.1.2 corrective maintenance: Reactive modification of a software product performed after delivery to correct discovered faults.

2.1.3 customer: The person, or persons, for whom the product is intended, and usually (but not necessarily) who decides the requirements.

2.1.4 emergency maintenance: Unscheduled corrective maintenance performed to keep a system operational.

2.1.5 interoperability testing: Testing conducted to ensure that a modified system retains the capability of exchanging information with systems of different types, and of using that information.

2.1.6 modification request (MR): A generic term that includes the forms associated with the various trouble/problem-reporting documents (e.g., incident report, trouble report) and the configuration change control documents [e.g., software change request (SCR), IEEE Std 1042-1987].

2.1.7 perfective maintenance: Modification of a software product after delivery to improve performance or maintainability.

2.1.8 project: A subsystem that is subject to maintenance activity.

2.1.9 regression test: Retesting to detect faults introduced by modification.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

2.1.10 repository: (A) A collection of all software-related artifacts (e.g., the software engineering environment) belonging to a system. (B) The location/format in which such a collection is stored.

2.1.11 reverse engineering: The process of extracting software system information (including documentation) from source code.

2.1.12 software maintenance: Modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.

2.1.13 system: A set of interlinked units organized to accomplish one or several specific functions.

2.1.14 user: The person or persons operating or interacting directly with the system.

2.2 Acronyms

The following acronyms are referred to in this standard:

SCM	Software configuration management
CSA	Configuration status accounting
FCA	Functional configuration audit
PCA	Physical configuration audit
PDL	Program design language
SQA	Software quality assurance
SCR	System/software change request
V&V	Verification and validation
VDD	Version description document

3. Software maintenance

This standard defines changes to software process through a defined maintenance process that includes the following phases:

a) Problem/modification identification and classification
b) Analysis
c) Design
d) Implementation
e) Regression/system testing
f) Acceptance testing
g) Delivery

These phases are graphically depicted in table 2. Software maintenance factors in table 3 are the entities qualified by the associated metrics/measures identified for each phase.

3.1 Problem/modification identification, classification, and prioritization

In this phase, software modifications are identified, classified, and assigned an initial priority ranking. Each modification request (MR) shall be evaluated to determine its classification and handling priority. Classification shall be identified from the following maintenance types: corrective, adaptive, perfective, and emergency. Metrics/measures and associated factors identified for this phase should be collected and reviewed at appropriate intervals (see table 3 and IEEE Stds 982.1-1988 and 982.2-1988).

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Table 2—Process model for software maintenance

	Problem identification	Analysis	Design	Implementation	System test	Acceptance test	Delivery
Input	MR	Project/system document Repository information Validated MR	Project/system document Source code Databases Analysis phase output	Source code Product/system document Results of design phase	Updated software documentation Test readiness review report Updated system	Test readiness review report Fully integrated system Acceptance test •Plans •Cases •Procedures	Tested/accepted system
Process	Assign change number Classify Accept or reject change Preliminary magnitude estimate Prioritize	Feasibility analysis Detailed analysis Redocument, if needed	Create test cases Revise •Requirements •Implementation plan	Code Unit test Test readiness review	Functional test Interface testing Regression testing Test readiness review	Acceptance test Interoperability test	PCA Install Training
Control	Uniquely identify MR Enter MR into repository	Conduct technical review Verify •Test strategy •Documentation is updated Identify safety & security issues	Software inspection/ review Verify design	Software inspection/review Verify •CM control of software •Traceability of design	Cm control of •Code •Listings •MR •Test documentation	Acceptance test Functional audit Establish baseline	PCA VDD
Output	Validated MR Process determinations	Feasability report Detailed analysis report Updated requirements Preliminary modification list Implementation plan Test strategy	Revised •Modification list •Detail analysis •Implementation plan Updated •Design baseline •Test plans	Updated •Software •Design documents •Test documents •User documents •Training material Test readiness review report	Tested system Test reports	New system baseline Acceptance test report FCA report	PCA report VDD
Metrics	See table 3						

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Table 3—Process model metrics for software maintenance

	Problem identification	Analysis	Design	Implementation	System test	Acceptance test	Delivery
Factors	Correctness Maintainability	Flexibility Traceability Reusability Usability Maintainability Comprehensibility	Flexibility Traceability Reusability Testability Maintainability Comprehensibility Reliability	Flexibility Traceability Maintainability Comprehensibility Reliability	Flexibility Traceability Verifiability Testability Interoperability Comprehensibility Reliability	Flexibility Traceability Interoperability Testability Comprehensibility Reliability	Completeness Reliability
Metrics	# of omissions on MR # of MR submittals # of duplicate MRs Time expended for problem validation	Requirement changes Documentation error rates Effort per function area (SQA, SE, etc.) Elapsed time (schedule) Error rates generated by priority & type	S/W complexity Design changes Effort per function area Elapsed time Test plans and procedure changes Error rates generated by priority and type Number of lines of code added, deleted, modified, tested Number of applications	Volume/functionality (function points or SLOC) Error rates generated by priority & type	Error rates by priority & type • Generated • Corrected	Error rates by priority & type • Generated • Corrected	Documentation changes (i.e., version description documents, training manuals, operation guidelines)

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

3.1.1 Input

Input for the problem/modification identification and classification phase shall be an MR.

3.1.2 Process

If a modification to the software is required, the following determinative activities must occur within the maintenance process:

a) Assign an identification number
b) Classify the type of maintenance
c) Analyze the modification to determine whether to accept, reject, or further evaluate
d) Make a preliminary estimate of the modification size/magnitude
e) Prioritize the modification
f) Assign MR to a block of modifications scheduled for implementation

Figure 2 summarizes the input, process, control, and output for the problem/modification identification and classification phase of maintenance. For additional information, see also A.4.1.

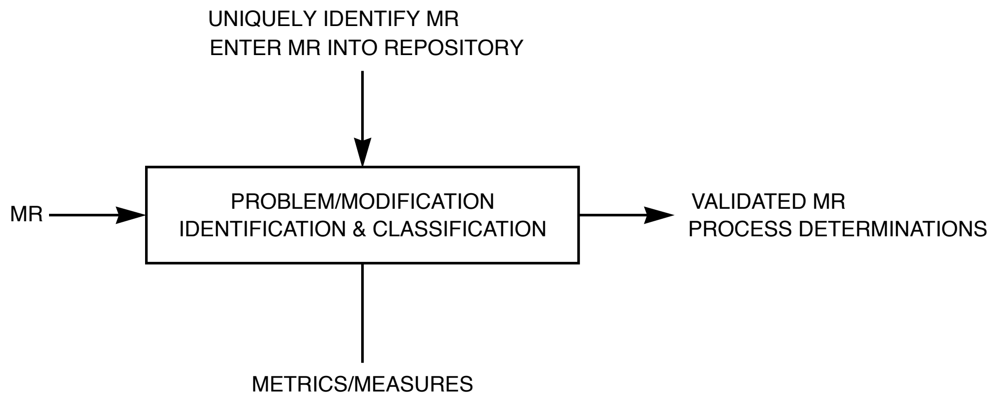

Figure 2—Problem/modification identification and classification phase

3.1.3 Control

MR and process determinations shall be uniquely identified and entered into a repository. See also A.11.1 for guidance.

3.1.4 Output

The output of this process shall be the validated MR and the process determinations that were stored in a repository. The repository shall contain the following items:

a) Statement of the problem or new requirement
b) Problem or requirement evaluation
c) Classification of the type of maintenance required
d) Initial priority
e) Verification data (for corrective modifications)
f) Initial estimate of resources required to modify the existing system

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

3.2 Analysis

The analysis phase shall use the repository information and the MR validated in the modification identification and classification phase, along with system and project documentation, to study the feasibility and scope of the modification and to devise a preliminary plan for design, implementation, test, and delivery. Metrics/measures and associated factors identified for this phase should be collected and reviewed at appropriate intervals (see table 3 and IEEE Stds 982.1-1988 and 982.2-1988).

Figure 3 summarizes the input, process, control, and output for the analysis phase of maintenance. For additional guidance, see A.4.2.

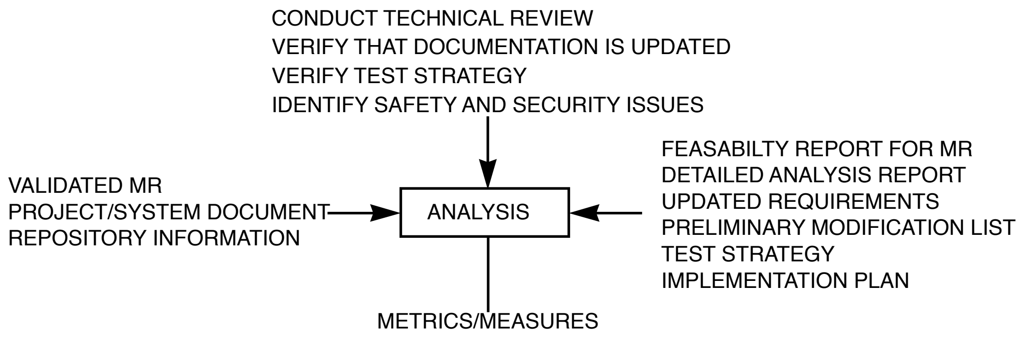

Figure 3—Analysis phase

3.2.1 Input

The input to the analysis phase of the maintenance process shall include the following:

a) Validated MR
b) Initial resource estimate and other repository information
c) Project and system documentation, if available

3.2.2 Process

Analysis is an iterative process having at least two components: (a) a feasibility analysis, and (b) a detailed analysis. If the documentation is not available or is insufficient and the source code is the only reliable representation of the software system, reverse engineering is recommended (see annex B for guidance).

3.2.2.1 Feasibility analysis

A feasibility analysis shall be performed for MR and a feasibility report (FR) shall be prepared. This FR should contain the following:

a) Impact of the modification
b) Alternate solutions, including protyping
c) Analysis of conversion requirements
d) Safety and security implications
e) Human factors
f) Short-term and long-term costs
g) Value of the benefit of making the modification

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

3.2.2.2 Detailed analysis

Detailed analysis shall provide the following:

a) Define firm requirements for the modification
b) Identify the elements of modification
c) Identify safety and security issues (see also A.9 and A.10 for guidance)
d) Devise a test strategy
e) Develop an implementation plan

In identifying the elements of modification (creating the preliminary modification list), analysts examines all products (e.g., software, specifications, databases, documentation) that are affected. Each of these products shall be identified, and generated if necessary, specifying the portions of the product to be modified, the interfaces affected, the user-noticeable changes expected, the relative degree and kind of experience required to make changes, and the estimated time to complete the modification.

The test strategy is based on input from the previous activity identifying the elements of modification. Requirements for at least three levels of test, including individual element tests, integration tests, and user-oriented functional acceptance tests shall be defined. Regression test requirements associated with each of these levels of test shall be identified as well. The test cases to be used for testing to establish the test baseline shall be revalidated.

A preliminary implementation plan shall state how the design, implementation, testing, and delivery of the modification is to be accomplished with a minimal impact to current users.

3.2.3 Control

Control of analysis shall include the following:

a) Retrieval of the relevant version of project and system documentation from the configuration control function of the organization
b) Review of the proposed changes and engineering analysis to assess technical and economic feasibility, and assess correctness
c) Identification of safety and security issues
d) Consideration of the integration of the proposed change within the existing software
e) Verification that all appropriate analysis and project documentation is updated and properly controlled
f) Verification that the test function of the organization is providing a strategy for testing the change(s), and that the change schedule can support the proposed test strategy
g) Review of the resource estimates and schedules and verification of their accuracy
h) Technical review to select the problem reports and proposed enhancements to be implemented in the new release. The list of changes shall be documented.

Consult A.6, A.7, and A.11.2 for guidance on activities related to verification and validation, software quality assurance, and software configuration management.

At the end of the analysis phase, a risk analysis shall be performed (see also A.8 for guidance). Using the output of the analysis phase, the preliminary resource estimate shall be revised, and a decision, that includes the customer, is made on whether to proceed to the design phase.

3.2.4 Output

The output of the maintenance process analysis phase shall include the following:

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

a) Feasibility report for MRs
b) Detailed analysis report
c) Updated requirements (including traceability list)
d) Preliminary modification list
e) Test strategy
f) Implementation plan

3.3 Design

In the design phase, all current system and project documentation, existing software and databases, and the output of the analysis phase (including detailed analysis, statements of requirements, identification of elements affected, test strategy, and implementation plan) shall be used to design the modification to the system. Metrics/measures and associated factors identified for this phase should be collected and reviewed at appropriate intervals (see table 3 and IEEE Stds 982.1-1988 and 982.2-1988).

Figure 4 summarizes the input, process, control, and output for the design phase of maintenance. (For additional guidance, see also A.4.3.)

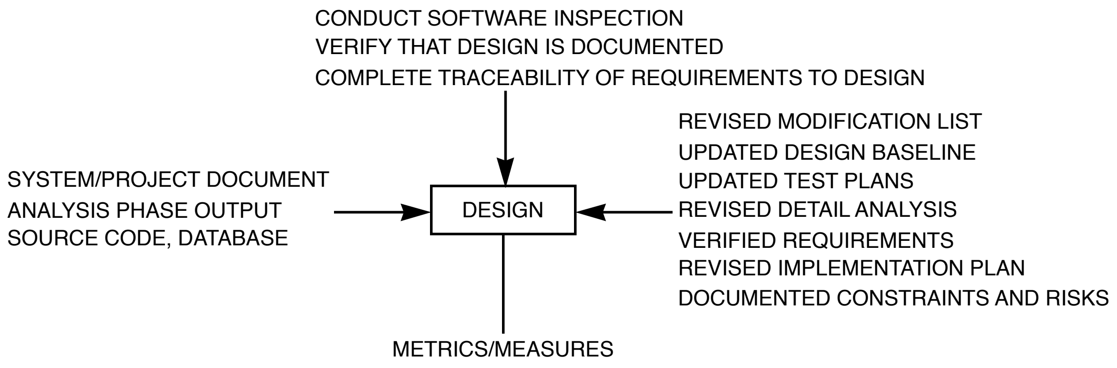
CONDUCT SOFTWARE INSPECTION
VERIFY THAT DESIGN IS DOCUMENTED
COMPLETE TRACEABILITY OF REQUIREMENTS TO DESIGN
SYSTEM/PROJECT DOCUMENT
ANALYSIS PHASE OUTPUT
SOURCE CODE, DATABASE
DESIGN
REVISED MODIFICATION LIST
UPDATED DESIGN BASELINE
UPDATED TEST PLANS
REVISED DETAIL ANALYSIS
VERIFIED REQUIREMENTS
REVISED IMPLEMENTATION PLAN
DOCUMENTED CONSTRAINTS AND RISKS
METRICS/MEASURES

Figure 4—Design phase

3.3.1 Input

Input to the design phase of the maintenance process shall include the following:

a) Analysis phase output, including
   1) Detailed analysis
   2) Updated statement of requirements
   3) Preliminary modification list
   4) Test strategy
   5) Implementation plan
b) System and project documentation
c) Existing source code, comments, and databases

3.3.2 Process

The process steps for design shall include the following:

a) Identifying affected software modules
b) Modifying software module documentation (e.g., data and control flow diagrams, schematics, PDL, etc.)

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

c) Creating test cases for the new design, including safety and security issues (for guidance, see also A.9 and A.10)
d) Identifying/creating regression tests
e) Identifying documentation (system/user) update requirements
f) Updating modification list

3.3.3 Control

The following control mechanism shall be used during the design phase of a change:

a) Conduct software inspection of the design in compliance with IEEE Std 1028-1988.
b) Verify that the new design/requirement is documented as a software change authorization (SCA), as per IEEE Std 1042-1987.
c) Verify the inclusion of new design material, including safety and security issues.
d) Verify that the appropriate test documentation has been updated.
e) Complete the traceability of the requirements to the design.

Consult A.6, A.7, and A.11.2 for guidance on activities related to verification and validation, software quality assurance, and software configuration management.

3.3.4 Output

The output of the design phase of the maintenance process shall include the following:

a) Revised modification list
b) Updated design baseline
c) Updated test plans
d) Revised detailed analysis
e) Verified requirements
f) Revised implementation plan
g) A list of documented constraints and risks (for guidance, see A.8)

3.4 Implementation

In the implementation phase, the results of the design phase, the current source code, and project and system documentation (i.e., the entire system as updated by the analysis and design phases) shall be used to drive the implementation effort. Metrics/measures and associated factors identified for this phase should be collected and reviewed at appropriate intervals (see table 3 and IEEE Stds 982.1-1988 and 982.2-1988).

Figure 5 summarizes the input, process, control, and output for the implementation phase of maintenance. For additional guidance, see also A.4.4.

3.4.1 Input

The input to the implementation phase shall include the following:

a) Results of the design phase
b) Current source code, comments, and databases
c) Project and system documentation

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

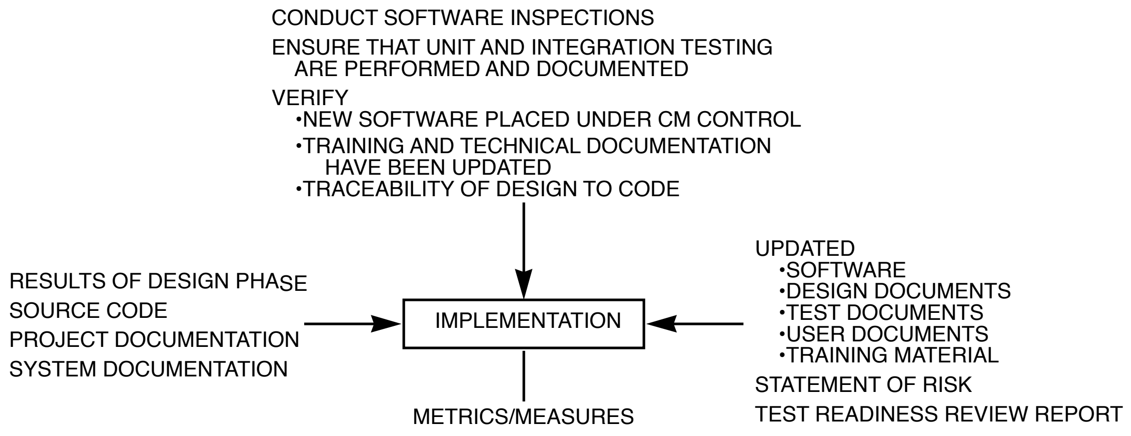
CONDUCT SOFTWARE INSPECTIONS
ENSURE THAT UNIT AND INTEGRATION TESTING ARE PERFORMED AND DOCUMENTED
VERIFY
•NEW SOFTWARE PLACED UNDER CM CONTROL
•TRAINING AND TECHNICAL DOCUMENTATION HAVE BEEN UPDATED
•TRACEABILITY OF DESIGN TO CODE
RESULTS OF DESIGN PHASE
SOURCE CODE
PROJECT DOCUMENTATION
SYSTEM DOCUMENTATION
IMPLEMENTATION
UPDATED
•SOFTWARE
•DESIGN DOCUMENTS
•TEST DOCUMENTS
•USER DOCUMENTS
•TRAINING MATERIAL
STATEMENT OF RISK
TEST READINESS REVIEW REPORT
METRICS/MEASURES

Figure 5—Implementation phase

3.4.2 Process

The implementation phase shall include the following four subprocesses, which may be repeated in an incremental, iterative approach:

a) Coding and unit testing
b) Integration
c) Risk analysis
d) Test readiness review

Metrics/measures and associated factors identified for this phase should be collected and reviewed at appropriate intervals (see table 3 and IEEE Stds 982.1-1988 and 982.2-1988).

3.4.2.1 Coding and unit testing

Implement the change into the code and perform unit testing and other appropriate SQA and V&V processes.

3.4.2.2 Integration

After the modifications are coded and unit-tested, or at appropriate intervals during coding, the modified software shall be integrated with the system and integration and regression tests shall be refined and performed. All effects (e.g., functional, performance, usability, safety), of the modification on the existing system shall be assessed. Any unacceptable impacts shall be noted. A return to the coding and unit-testing subprocess shall be made to remedy these.

3.4.2.3 Risk analysis and review

In the implementation phase, risk analysis and review shall be performed periodically during the phase rather than at its end, as in the design and analysis phases. Metrics/measurement data should be used to quantify risk analysis. For additional guidance, see A.8.

3.4.2.4 Test-readiness review

To assess preparedness for system test, a test-readiness review shall be held in accordance with IEEE Std 1028-1988.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

3.4.3 Control

The control of implementation shall include the following:

a) Conduct software inspections of the code in compliance with IEEE Std 1028-1988.
b) Ensure that unit and integration testing are performed and documented in a software development folder.
c) Ensure that test documentation (e.g., test plan, test cases, and test procedures) are either updated or created.
d) Identify, document, and resolve any risks exposed during software and test-readiness reviews.
e) Verify that the new software is placed under software configuration management control.
f) Verify that the training and technical documentation have been updated.
g) Verify the traceability of the design to the code.

Consult A.6, A.7, and A.11.2 for guidance on activities related to verification and validation, software quality assurance, and software configuration management.

3.4.4 Output

The output of the implementation phase shall include the following:

a) Updated software
b) Updated design documentation
c) Updated test documentation
d) Updated user documentation
e) Updated training material
f) A statement of risk and impact to users
g) Test readiness review report (see IEEE Std 1028-1988)

3.5 System test

System testing, as defined in IEEE Std 610.12-1990, shall be performed on the modified system. Regression testing is a part of system testing and shall be performed to validate that the modified code does not introduce faults that did not exist prior to the maintenance activity. Metrics/measures and associated factors identified for this phase should be collected and reviewed at appropriate intervals (see table 3 and IEEE Stds 982.1-1988 and 982.2-1988).

Figure 6 summarizes the input, process, control, and output for the system test phase of maintenance. For additional guidance, see also A.4.5.

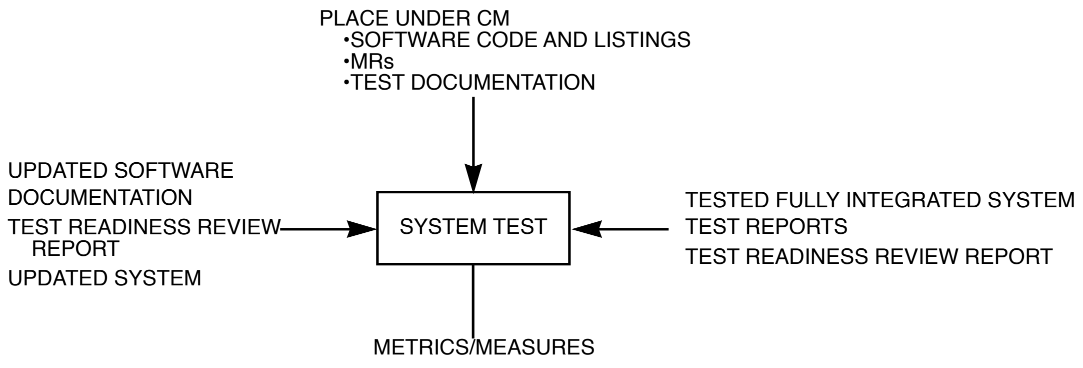

Figure 6—System test phase

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

3.5.1 Input

Input to the system test phase of maintenance shall include the following:

a) Test-readiness review report
b) Documentation, which includes:
    1) System test plans (IEEE 829-1983)
    2) System test cases (IEEE 829-1983)
    3) System test procedures (IEEE 829-1983)
    4) User manuals
    5) Design
c) Updated system

3.5.2 Process

System tests shall be conducted on a fully integrated system. Testing shall include the performance of

a) System functional test
b) Interface testing
c) Regression testing
d) Test readiness review to assess preparedness for acceptance testing

NOTE—Results of tests conducted prior to the test-readiness review should not be used as part of the system test report to substantiate requirements at the system level. This is necessary to ensure that the test organization does not consider that testing all parts (one at a time) of the system constitutes a "system test."

3.5.3 Control

System tests shall be conducted by an independent test function, or by the software quality assurance function. Prior to the completion of system testing, the test function shall be responsible for reporting the status of the criteria that had been established in the test plan for satisfactory completion of system testing. The status shall be reported to the appropriate review committee prior to proceeding to acceptance testing. Software code listings, MRs and test documentation shall be placed under SCM. The customer shall participate in the review to ascertain that the maintenance release is ready to begin acceptance testing.

Consult A.6, A.7, and A.11.2 for guidance on activities related to verification and validation, software quality assurance, and SCM.

3.5.4 Output

The output for this phase of maintenance shall include the following:

a) Tested and fully integrated system
b) Test report
c) Test readiness review report

3.6 Acceptance test

Acceptance tests shall be conducted on a fully integrated system. Acceptance tests shall be performed by either the customer, the user of the modification package, or a third party designated by the customer. An acceptance test is conducted with software that is under SCM in accordance with the provisions of IEEE Std 828-1990, and in accordance with the IEEE Std 730-1989. Acceptance testing, as defined in IEEE Std 610.12-1990, shall be performed on the modified system. Metrics/measures and associated factors

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

identified for this phase should be collected and reviewed at appropriate intervals (see table 3 and IEEE Stds 982.1-1988 and 982.2-1988).

Figure 7 summarizes the input, process, control, and output for the acceptance test phase of maintenance. For additional guidance, see also A.4.6.

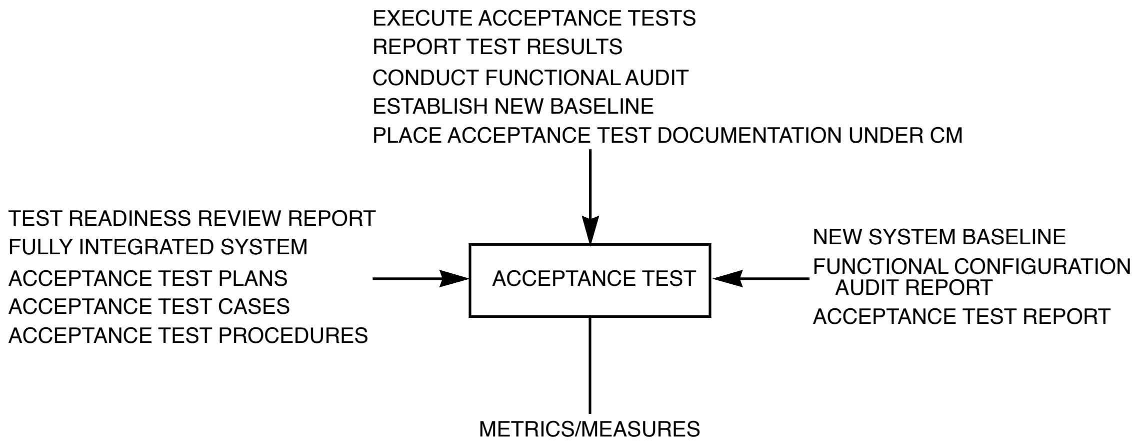

Figure 7—Acceptance test phase

3.6.1 Input

The input for acceptance testing shall include the following:

a) Test readiness review report
b) Fully integrated system
c) Acceptance test plans
d) Acceptance test cases
e) Acceptance test procedures

3.6.2 Process

The following are the process steps for acceptance testing:

a) Perform acceptance tests at the functional level
b) Perform interoperability testing
c) Perform regression testing

3.6.3 Control

Control of acceptance tests shall include the following:

a) Execute acceptance tests
b) Report test results for the functional configuration audit (FCA)
c) Conduct functional audit
d) Establish the new system baseline
e) Place the acceptance test documentation under SCM control

Consult A.6, A.7, and A.11.2 for guidance on activities related to verification and validation, software quality assurance, and SCM.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

3.6.4 Output

The output for the acceptance phase shall include the following:

a) New system baseline
b) Functional configuration audit report (see IEEE Std 1028-1988)
c) Acceptance test report (see IEEE Std 1042-1987)

NOTE—The customer shall be responsible for the acceptance test report.

3.7 Delivery

This subclause describes the requirements for the delivery of a modified software system. Metrics/measures and associated factors identified for this phase should be collected and reviewed at appropriate intervals (see table 3 and IEEE Stds 982.1-1988 and 982.2-1988).

Figure 8 summarizes the input, process, control, and output for the delivery phase of maintenance. For additional guidance, see also A.4.7.

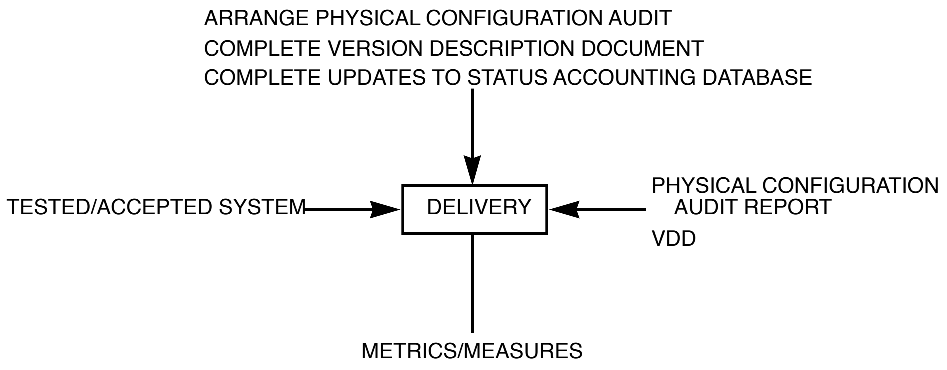
ARRANGE PHYSICAL CONFIGURATION AUDIT
COMPLETE VERSION DESCRIPTION DOCUMENT
COMPLETE UPDATES TO STATUS ACCOUNTING DATABASE
TESTED/ACCEPTED SYSTEM
DELIVERY
PHYSICAL CONFIGURATION AUDIT REPORT
VDD
METRICS/MEASURES

Figure 8—Delivery phase

3.7.1 Input

The input to this phase of the maintenance process shall be the fully tested version of the system as represented in the new baseline.

3.7.2 Process

The process steps for delivery of a modified product shall include the following:

a) Conduct a physical configuration audit
b) Notify the user community
c) Develop an archival version of the system for backup
d) Perform installation and training at the customer facility

3.7.3 Control

Control for delivery shall include the following:

a) Arrange and document a physical configuration audit
b) Provide system materials for access to users, including replication and distribution

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

c) Complete the version description document (IEEE Std 1042-1987)
d) Complete updates to status accounting database
e) Place under SCM control

Consult A.6, A.7, and A.11.2 for guidance on activities related to verification and validation, software quality assurance, and SCM.

3.7.4 Output

Output for delivery shall include the following:

a) PCA report (IEEE Std 1028-1988)
b) Version description document (VDD)

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Annexes

(These informative annexes are not a part of IEEE Std 1219-1992, IEEE standard for software maintenance, but are included for information only.)

Annex A
Maintenance Guidelines

(informative)

A.1 Definitions

The definitions listed below define terms as used in this annex.

A.1.1 completeness: The state of software in which full implementation of the required functions is provided.

A.1.2 comprehensibility: The quality of being able to be understood; intelligibility, conceivability.

A.1.3 consistency: Uniformity of design and implementation techniques and notation.

A.1.4 correctness: The state of software in which traceability, consistency, and completeness are provided.

A.1.5 instrumentation: The attributes of software that provide for the measurement of usage or identification of errors.

A.1.6 modularity: Being provided with a structure of highly independent modules.

A.1.7 preventive maintenance: Maintenance performed for the purpose of preventing problems before they occur.

A.1.8 safety: The ability of a system to avoid catastrophic behavior.

A.1.9 self-descriptiveness: The extent of a software's ability to provide an explanation of the implementation of a function or functions.

A.1.10 simplicity: The provision of implementation of functions in the most understandable manner (usually avoidance of practices that increase complexity).

A.1.11 testability: The ability of a software to provide simplicity, modularity, instrumentation, and self-descriptiveness.

A.1.12 traceability: The ability of a software to provide a thread from the requirements to the implementation, with respect to the specific development and operational environment.

A.1.13 verifiability: The capability of a software to be verified, proved, or confirmed by examination or investigation.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

A.2 References

The following standards are directly referenced in this annex. Table 1 provides a cross-reference of IEEE standards that address various topics related to software maintenance. These standards are binding to the extent specified within the text of this standard and are referenced to avoid duplication of requirements.

IEEE Std 730-1989, IEEE Standard for Software Quality Assurance Plans (ANSI).[4]

IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans (ANSI).

IEEE Std 829-1983 (Reaff 1991), IEEE Standard for Software Test Documentation (ANSI).

IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software (ANSI).

IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software (ANSI).

IEEE Std 983-1986 (w1992), IEEE Guide for Software Quality Assurance Planning (ANSI).[5]

IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans (ANSI).

IEEE Std 1028-1988, IEEE Standard for Software Reviews and Audits (ANSI).

IEEE Std 1042-1987, IEEE Guide to Software Configuration Management (ANSI).

IEEE Std 1058.1-1987, IEEE Standard for Software Project Management Plans (ANSI).

NBS Special Publication 500-106, Guidance on Software Maintenance.[6]

P1228, Software Safety Plans, Draft H, April 27, 1992.[7]

A.3 Maintenance planning

Planning for maintenance may include: determining the maintenance effort, determining the current maintenance process, quantifying the maintenance effort, projecting maintenance requirements, and developing a maintenance plan. IEEE Std 1058.1-1987 may also be used for guidance in maintenance planning.

A.3.1 Determine maintenance effort

The first step in the maintenance planning process is an analysis of current service levels and capabilities. This includes an analysis of the existing maintenance portfolio and the state of each system within that portfolio. At the system level, each system should be examined to determine the following:

[4]IEEE publications are available from the Institute of Electrical and Electronics Engineers, Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

[5]IEEE Std 983-1986 has been withdrawn; however, copies can be obtained from the IEEE Standards Department, IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

[6]NBS publications are available from the Superintendent of Documents, US Government Printing Office, P.O. Box 37082, Washington, DC 20013-7082, USA.

[7]This authorized standards project was not approved by the IEEE Standards Board at the time IEEE Std 1219-1992 went to press. It is available from the IEEE Service Center, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

a) Age since being placed in production
b) Number and type of changes during life
c) Usefulness of the system
d) Types and number of requests received for changes
e) Quality and timeliness of documentation
f) Any existing performance statistics (CPU, disk I/O, etc.)

Descriptions at the portfolio level can assist in describing the overall effort and needs of the maintenance area. This includes the amount and kinds of functional system overlap and gaps within the portfolio architecture.

The reviews of the maintenance staff and the maintenance procedures are also necessary to determine the overall maintenance effort. The analysis at this stage is simply to gather those measures needed to determine the following:

a) The number of maintainers, their job descriptions, and their actual jobs
b) The experience level of the maintenance staff, both industry-wide and for the particular application
c) The rate of turnover and possible reasons for leaving
d) Current written maintenance methods at the systems and program level
e) Actual methods used by programming staff
f) Tools used to support the maintenance process and how they are used

Information at this stage is used to define the baseline for the maintenance organization and provide a means of assessing the necessary changes.

A.3.2 Determine current maintenance process

The maintenance process is a natural outgrowth of many of the baseline measures. Once those measures have been collected, the actual process needs to be determined. In some organizations, the process is tailored to the type of maintenance being performed and can be divided in several different ways. This can include different processes for corrections vs. enhancements, small changes vs. large changes, etc. It is helpful to classify the maintenance approaches used before defining the processes.

Each process will then be described by a series of events. In general, the flow of work is described from receipt of a request to its implementation and delivery.

A.3.3 Quantify maintenance effort

Each step in the process needs to be described numerically in terms of volumes or time. These numbers can then be used as a basis to determine the actual performance of the maintenance organization.

A.3.4 Project maintenance requirements

At this stage, the maintenance process needs to be coupled to the business environment. A review of future expectations should be completed and may include

a) Expected external or regulatory changes to the system
b) Expected internal changes to support new requirements
c) Wish-list of new functions and features
d) Expected upgrades for performance, adaptability, connectivity, etc.
e) New lines of business that need to be supported
f) New technologies that need to be incorporated

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

These need to be quantified (or sized) to determine the future maintenance load for the organization.

A.3.5 Develop maintenance plan

The information collected will provide a basis for a new maintenance plan. The plan should cover four main areas:

a) Maintenance process
b) Organization
c) Resource allocations
d) Performance tracking

Each of these issues are addressed and imbedded in the final maintenance plan. The actual process should be described in terms of its scope, the sequence of the process, and the control of the process.

A.3.5.1 Process scope

The plan needs to define the boundaries of the maintenance process. The process begins at some point (receipt of the request) and will end with some action (delivery and sign-off). In addition, the difference between maintenance and development should be addressed at this point. Is an enhancement considered to be a new development, or maintenance? At what point does a newly developed system enter the maintenance process?

Another issue that should be defined within the scope is whether and how the maintenance process will be categorized. Will there be differences between reporting and other types of maintenance? Will adaptations and enhancements be considered within the same process or will they be handled differently?

A.3.5.2 Process sequence

The overall flow of work (and paper) needs to be described. This should include the following:

a) Entry into automated SCM and project management systems
b) Descriptions of each process step and their interfaces
c) The data flow between processes

The sequence should use the process described in this standard as a guideline.

A.3.5.3 Process control

Each step in the process should be controlled and measured. Expected levels of performance should be defined. The control mechanisms should be automated, if possible. The control process should follow the standards set forth in this document.

A.3.5.4 Organization

Staff size can be estimated from the current work load and estimates of future needs. This estimate may also be based on the expected productivity of each step in the process.

A.3.5.5 Resource allocation

An important part of the maintenance plan is an analysis of the hardware and software most appropriate to support the organization's needs. The development, maintenance, and target platforms should be defined and differences between the environments described. Tool sets that enhance productivity should be identified

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

and provided. The tools should be accessible by all who need them, and sufficient training should be provided so that their use is well understood.

A.3.5.6 Tracking

Once the process is in place, it should be tracked and evaluated to judge its effectiveness. If each step in the process has measurement criteria, it should be a straightforward process to collect and evaluate performance over time.

A.3.5.7 Implementation of plan

Implementing a maintenance plan is accomplished in the same way that any organizational change is performed. It is important to have as much technical, professional, and managerial input as possible when the plan is being developed.

A.4 Maintenance process

The phases presented herein mirror those in the standard.

A.4.1 Problem/modification identification and classification

Each system/software and modification request should be evaluated to determine its classification and handling priority and assignment for implementation as a block of modifications that will be released to the user. A suggested method for this is to hold a periodic review of all submitted items. This provides a regular, focused forum and helps prevent the review/analyze/design/implement/test process (which may be iterative) from stalling due to lack of direction. It also increases awareness of the most requested and most critical items. An agenda should be distributed prior to the meeting listing the items to be classified and prioritized.

The frequency and duration for modification classification review meetings should be project-dependent. A guideline might be to consider them as status reviews rather than as technical reviews. Using this guideline, if, after the first few sessions, the reviews take more than an hour or so, their frequency should be increased to as often as weekly. If review time still seems insufficient, determine whether one of the following cases applies and handle accordingly:

a) Discussion is focused on major system enhancements (perfective maintenance). This may require analysis/review cycles of a development magnitude, rather than at a sustaining maintenance level.
b) The system is new, and design/implementation problems require a significant maintenance effort immediately following delivery. A suggested strategy, to be used where the impact to operations is acceptable, is to classify the MRs as corrective/adaptive/perfective/preventive and integrate them into sets that share the same design areas. Then, rather than prioritizing by individual MR, prioritize by set. This minimizes repetition of design, code, test, and delivery tasks for the same code modules. In this case, review meetings may be longer and more frequent until the system is stabilized. A long-term plan should have a goal of gradually reducing the review meeting overhead.
c) The system is aging, and system replacement or reverse engineering and redesign are under consideration.

These are not the only possible cases. They are described to highlight the importance of understanding the goals for classifying modifications as they apply to a particular system.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Modification requests should be assigned a priority from

a) The MR author or a designated representative
b) A knowledgeable user
c) A domain expert
d) Software engineers (depending on the project, this may include representatives of system analysis and design, development, integration, test, maintenance, quality control, and SCM), or
e) Project management

The following criteria should be considered:

a) Rough resource estimate, which may be derived from
   1) Relative ease/difficulty of implementation
   2) Approximate time to implement given available human and system resources
b) Expected impact to current and future users
   1) General statement of benefits
   2) General statement of drawbacks
c) Assignment to an implementation of a block of modifications that are scheduled to minimize the impact on the user

Thorough, low-level cost and impact studies should be performed in the analysis phase, but general cost estimates and statements of impact are desirable for initial classification. Since this is an iterative process, it is likely that handling priority may change during the phases that follow.

A.4.2 Analysis

A modification request may generate several system-level functional, performance, usability, reliability, and maintainability requirements. Each of these may be further decomposed into several software, database, interface, documentation, and hardware requirements. Involvement of requesters, implementers, and users is necessary to ensure that requirement(s) are unambiguous statements of the request.

Software change impact analysis should

a) Identify potential ripple effects
b) Allow trade-offs between suggested software change approaches to be considered
c) Be done with the help of documentation abstracted from the source code
d) Consider the history of prior changes, both successful and unsuccessful

The following should be included in a detailed analysis:

a) Determine if additional problem analysis/identification is required
b) Record acceptance or rejection of the proposed change(s)
c) Develop an agreed-upon project plan
d) Evaluate any software or hardware constraints that may result from the changes and that need consideration during the design phase
e) Document any project or software risks resulting from the analysis to be considered for subsequent phases of the change life cycle
f) Recommend the use of existing designs, if applicable

A.4.3 Design

Actual implementation should begin during this phase, while keeping in mind the continued feasibility of the proposed change. For example, the engineering staff may not fully understand the impact and magnitude

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

of a change until the design is complete, or the design of a specific change may be too complex to implement.

The vehicles for communicating the updated design are project/organization-dependent and may include portions of a current design specification, software development files, and entries in software engineering case tool databases. Other items that may be generated during this phase include a revised analysis, revised statements of requirements, a revised list of elements affected, a revised plan for implementation, and a revised risk analysis.

The specifics of the design process may vary from one project to the next and are dependent on such variables as tool use, size of modification, size of existing system, availability of a development system, and accessibility to users and requesting organizations. Product characteristics should also be evaluated when developing the design so that decisions on how modules of software will be changed will take into consideration the reliability and future maintainability of the total system, rather than focusing on expediency.

A.4.4 Implementation

The primary inputs to this phase are the results of the design phase. Other inputs required for successful control of this phase include the following:

a) Approved and controlled requirements and design documentation
b) An agreed-upon set of coding standards to be used by the maintenance staff
c) Any design metrics/measurement that may be applicable to the implementation phase (these metrics/measures may provide insight into code that may be complex to develop or maintain)
d) A detailed implementation schedule, noting how many code reviews will take place and at what level
e) A set of responses to the defined risks from the previous phase that are applicable to the testing phase

Risk analysis and review should be performed periodically during this phase rather than at its end, as in the design and analysis phases. This is recommended because a high percentage of design, cost, and performance problems and risks are exposed while modifying the system. Careful measurement of this process is necessary, and becomes especially important if the number of iterations of the coding and unit testing and integration subprocesses is out of scope with the modification. If this is found true, the feasibility of the design and/or modification request may need reassessment, and a return to the design, analysis, or even the modification identification and classification phase may be warranted.

Prior to a review, the following information may be prepared and provided to attendees:

a) Entry criteria for system test
b) Resource allocation and need schedule
c) Detailed test schedule
d) Test documentation forms
e) Anomaly resolution procedures
f) SCM procedures
g) Exit criteria for system test
h) Lower-level test results

Attendees may include the following:

a) The MR author, a designated representative, or a knowledgeable user
b) A domain expert

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

c) Software engineers (depending on the project, this may include representatives of system analysis and design, development, integration, test, maintenance, quality control, and SCM)
d) Project management

A.4.5 System test

It is essential to maintain management controls over the execution of system tests to ensure that the non-technical issues concerning budget and schedule are given the proper attention. This function also ensures that the proper controls are in place to evaluate the product during testing for completeness and accuracy.

System tests should be performed by an independent test organization, and may be witnessed by the customer and the end-user. System test is performed on software that is under SCM in accordance with the provisions of IEEE Std 828-1990.[8] Software quality assurance is conducted in accordance with the provisions of IEEE Std 730-1989. The system test is performed with as complete a system as is possible, using simulation/stimulation to the smallest degree possible. Functions are tested from input to output.

For maintenance releases, it is possible that other testing may have to be done to satisfy requirements to interact with other systems or subsystems. It may also be necessary to conduct testing to validate that faults are not introduced as a result of changes.

System tests should be conducted on a fully integrated system. Simulation/stimulation may be used in cases where it is not possible to have the completely integrated system in the test facility. However, its use should be minimized. If utilized, it should be identified and evaluated/justified.

The organization that is responsible for system tests should be independent of the software developers and designers, but these organizations may be used as a resource of test personnel. Control of software builds, and all pertinent files (source, object, libraries, etc.) during a system test should be done by the SCM function. Controls to ensure product integrity are executed by the software quality assurance function. They should ensure that the changes to the products that are submitted are in fact authorized and technically correct.

If changes have been made to the software, or test cases, since the software has been delivered, then it may be necessary to run regression and unit tests during the analysis phase in order to establish the product baseline.

A.4.6 Acceptance test

The acceptance test is performed to ensure that the products of the modification are satisfactory to the customer. The products include the software system and the documentation necessary to support it. The culmination of the acceptance test is usually the completion of a functional audit and a physical audit (see IEEE Std 1028-1988).

For maintenance releases, it is possible that other testing may have to be done to satisfy requirements to interact with other systems or subsystems. It may also be necessary to conduct testing to validate that faults are not introduced as a result of changes.

Acceptance tests should be conducted on a fully integrated system. Simulation and/or stimulation may be used in cases where it is not possible to have the completely integrated system in the test facility. However, its use should be minimized. This requirement may be modified to include the case where lower-level testing is performed. The customer, or the customer's representative, is responsible for determining the facility

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

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

requirements that are necessary. These requirements are documented by the developer in the modification plan. Acceptance test facilities may be provided by either the developer or the customer, or a combination of both.

Results of tests conducted prior to the acceptance test readiness review may be used by the customer to reduce the scope of acceptance tests. If this is done, the customer should document, in the acceptance test report, which results were taken from previous tests.

Prior to the completion of acceptance testing, the test organization should be responsible for reporting the status of the criteria that had been established in the test plan for satisfactory completion of acceptance testing. The status should be reported to the appropriate review committee. The customer, or the customer's representative, should chair the review group and evaluate the exit criteria to ensure that the maintenance release is ready for delivery to the end-user.

A.4.7 Delivery

Based on how the users access the system, the delivery may entail replacing the existing system with the new version, duplication of the configuration controlled master for delivery to remote users, or digital transmission.

To reduce the risks associated with installation of the new version of a software system, project management should plan for and document alternative installation procedures that may insure minimal impact on the system users due to unforeseen software failures not detected during testing. The planning should address time-critical factors (e.g., date/times available for installation, critical milestones of the users, etc.) and restoration/recovery procedures.

When a system modification affects user interfaces or is a significant modification of functionality, user training may be necessary. This can include formal (classroom) and non-formal methods. When the modifications result in significant documentation changes, user training should be considered.

SCM is responsible for backing up the system. To ensure recovery, the backup should consist of the existing system version as well as the new version. To facilitate disaster recovery, complete copies of the system backup should be archived at an off-site location. The backup should consist of source code, requirement documentation, design documentation, test documentation (including test case data), and the support environment [i.e., operating system, compiler, assembler, test driver(s), and other tools].

A.5 Maintenance forms

Recording, tracking, and implementing software maintenance requires that various forms be completed and managed. What follows is a list of forms that may be used to perform maintenance, and the IEEE standard that explains their format and usage.

a) Test Log—IEEE Std 829-1983
b) Test Incident Report—IEEE Std 829-1983
c) Test Summary Report—IEEE Std 829-1983
d) Test Design Specification—IEEE Std 829-1983
e) System/Software Change Request—IEEE Std 1042-1987
f) Software Change Authorization—IEEE Std 1042-1987

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

A.6 Verification and validation (V&V)

IEEE Std 1012-1986 should be used to verify and validate that all maintenance requirements are met.

A.7 Software quality assurance

Software quality assurance should be considered when any modifications are made to an existing system. A modification to one segment of the system can cause errors to appear somewhere else. Other concerns can include: version control, new documentation release, etc. To ensure that quality is maintained for all modifications, standards as stated in IEEE Std 730-1989, and IEEE Std 983-1986 should be adhered to.

Continuing product quality assurance includes the following:

a) Adherence to the maintenance plan and approach
b) Testing (including regression testing)
c) Revalidation activities
d) Recertification activities

The same types and levels of assurance (e.g., inspections, reviews, audits, verification and validation, evaluation of metric data, records) should be conducted as were performed during development; the degree and conduct of these activities is specified in the software maintenance plan. Special care should be given to ensuring that the original system documentation continues to describe the actual product; during operational use, the time criticality of effecting a repair to the product often results in the lack of related changes to the documentation, with concomitant loss of configuration control. Similarly, the operational facility should be able to maintain the distinction between proposed fixes to the software product needed to provide an immediate resolution of a critical problem; adopted fixes that, due to time-criticality, should be utilized prior to having been authorized; and those corrections that, through testing, revalidation, and recertification, have been officially authorized.

A.8 Risk assessment

Software maintenance activities consume considerable resources to implement software changes in existing systems. Traditionally, systems are tested to detect defects in the software. Since defects in various software workproducts cause failures with different consequences, the significance (or risk) of each failure/defect varies.

Software risk is defined as the potential loss due to failure during a specific time period. Risk is measured as a product of the frequency or likelihood of loss and the magnitude or level of exposure. Risk assessment begins with an analysis of external exposure—determination of the magnitude of loss that can result from invalid actions. The external exposure is mapped onto the system to determine the magnitude of loss caused by faults in individual software workproducts. The likelihood of failure for each workproduct is based on its use, verification, validation, adaptability, and size characteristics.

In the context of maintenance, the failure can be product- or process-oriented. That is, the failure of the product (i.e., errors in function and performance) and process (i.e., inaccurate estimates) have the potential of increasing costs of the product and are therefore considered risks. Risk abatement techniques for product risks include testing and maintainability measurement. Risk abatement for the process includes software change impact analysis.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

To measure software risk, several functions should be performed: external exposure identification, structural exposure analysis, and software failure likelihood estimation. The following is an outline of these functions as they pertain to software maintenance risk assessment.

A.8.1 External exposure identification

The external exposure identification function has two primary objectives. The first is to determine what actions in the environment outside of the software can contribute to loss. The second objective is to assess the significance of the loss.

To this end, the procedure involves the following steps:

a) *Definition of environmental hazards*. Determine how the organization intends to use the software system, and potential hazards such as financial troubles, explosions, incorrect patient treatment, misinformation, loss of life, and like accidents.
b) *Identification of accident sequences*. Investigate how accidents can occur and record them in event trees, scenarios, or annotated event-sequence diagrams.
c) *Failure mode analysis*. Identify failure modes from accident sequences and record them in fault trees. Key failure areas are identified by working backwards in the fault tree.
d) *Consequence analysis*. Determine the consequence of the faults by weighting the loss estimated for each accident scenario. Since this has a wide range of factors and conditions, care should be taken to focus on the key failures.

A.8.2 Structural exposure analysis

Structural exposure analysis is performed to discover how and where software faults can contribute to losses identified in the external exposure assessment. The objective of this function is to assign exposure levels to individual workproducts based on their capability to cause failures. The procedure includes the following activities:

a) *Identify software failure modes*. Indicate where erroneous information can contribute to loss and investigate how the software can fail as a result.
b) *Determine workproduct fault potential*. Locate the potential faults related to the fault modes and identify associated relationships between faults and losses.
c) *Analyze mode use*. Locate where potentially faulty workproducts are used.
d) *Compute workproduct exposure*. Estimate the module exposure by summing its potential loss for all accident scenarios to which it is related.

A.8.3 Software failure likelihood

The objective of the software failure likelihood function is to predict failure likelihood from workproduct and maintenance process characteristics. As software testing proceeds, information about the testing process is used to update or confirm our initial estimates of failure likelihood. The likelihood of software failure depends on the number of faults in a workproduct and the probability that the fault will be encountered in the operation of the system. That is, failure likelihood depends on the number of faults and the probability that the faults will cause failures. Risks for each workproduct is determined by the probability of each type of failure times the costs of that failure.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

A.9 Safety

Safety is the ability of a system to avoid catastrophic behavior. Safety requirements may identify critical functions whose failure may be hazardous to people or property. The results of the procedure described in A.8.1 may also be applicable. Draft standard P1228 contains information that may be used to create and maintain systems.

A.10 Security

The degree to which the system and information access needs to be protected can have an effect on the manner in which the system is maintained. A system is secure if unauthorized personnel cannot get at protected information and to the system itself.

Security during the maintenance process should ensure the following:

a) The integrity of the system is preserved by ensuring that only authorized personnel have access to the system and only authorized changes are implemented. This is accomplished in cooperation with SCM and SQA.
b) Security features implemented during system development are not compromised, either by inadvertent action or failure to comply with the existing security requirements.
c) New functions added to the system are compatible with the existing security features.

A.11 Software configuration management (SCM)

Software configuration management (SCM) is a critical element of the software maintenance process. Conformance to IEEE Std 828-1990 and IEEE Std 1042-1987 should be adhered to. SCM, a procedure-driven process, depends on sound, workable, and repeatable procedures to implement the software engineering release function. The procedures should provide for the verification, validation, and certification of each step required to identify, authorize, implement, and release the software product. These procedures should also define the methods used to track the change throughout the maintenance process, to provide the required traceability, to ensure the integrity of the product, to keep project management informed with periodic reports, and to document all the activities involved. During the implementation of the software engineering release process, SCM has responsibilities in each of the major categories of SCM; i.e., configuration identification, configuration audits, change control, and status accounting.

SCM of documentation during system test should be done by the SCM function. The documents to be managed are

a) System documentation
b) Software code and listings
c) MRs
d) Test documentation

Although SCM is not very involved during the initial phases of traditional software development, SCM should be actively pursued throughout the entire software maintenance process. Failure to provide rigorous SCM can result in chaos during the maintenance process. The following items address the SCM requirements that should prevail during each phase of the maintenance process as defined in this standard.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

A.11.1 Problem/modification identification and classification

The SCM process is the principal element of the problem identification phase of software maintenance. SCM is responsible for receiving and logging the problem (corrective, adaptive, corrective) into the configuration status accounting (CSA) system (IEEE Std 1042-1987). SCM personnel are responsible for routing the problem to the proper personnel (e.g., system/software engineering, test) for validation and evaluation of the problem. SCM provides tracking and coordination of the problem documentation during this phase.

A.11.2 Analysis

During the analysis phase, it is an SCM responsibility to provide up-to-date documentation to the personnel performing the analysis (e.g., systems/software engineering). SCM should also provide up-to-date listings of the source code and accurate CSA reports showing the current status of the problem.

At the completion of the analysis phase, it is an SCM responsibility to ensure the analysis results are presented to a review board (an SCM activity). The review board provides visibility and tracking into the problem resolution since it is responsible for authorizing further analysis or implementation. The review board assigns the problems approved for implementation to a software engineering release (block change) package. Although each problem is implemented independently, control is done on a block of problems associated via the software engineering release. SCM is responsible for documenting the proceedings of the review board and updating the CSA records associated with the problem.

A.11.3 Design

During the design phase, SCM is responsible for ensuring that the design personnel have up-to-date documentation from the system library. It is crucial that design personnel receive any changes to documentation as soon as possible after receipt. It is an SCM responsibility to archive and safeguard software inspection/review results and other design data provided by the design personnel. In an automated design environment (e.g., computer-aided software engineering [CASE]), SCM may be required to provide assistance to the design personnel in maintaining version control. SCM is expected to provide assistance and guidance to design personnel in the selection and usage of consistent naming conventions.

At the completion of the design phase, it is a configuration management responsibility to ensure that design documentation is placed in safekeeping and receives SCM control to protect the integrity of the design product. In automated systems, configuration management is responsible for ensuring rigorous version control.

A.11.4 Implementation

During implementation, SCM is responsible for providing the programmers with copies of the modules to be changed and ensuring rigorous version control. In situations where multiple changes are made to a single module, SCM is responsible for ensuring proper staging to prevent losing changes. This requires complete cooperation and coordination between SCM and the maintenance team. SCM should notify the maintenance team when updated requirement or design data becomes available.

SCM is responsible for maintaining configuration control over all the support tools used in the software maintenance process. Control of the tools (i.e., compilers, assemblers, operating systems, link editors) is crucial to avoid major impacts to the maintenance schedule and in preventing unnecessary re-work.

At the completion of the implementation phase, SCM is responsible for collecting all the modules that have been changed and placing them in secure library facilities. If a software development folder concept is used, SCM should ensure that each is placed under configuration control. SCM is often responsible for regenerating the system (compile, link, etc.) and making it available for the test personnel. This is a good practice that

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

ensures consistent results and products for the test group. The librarian of a chief programmer team may perform this SCM task.

SCM is responsible for ensuring that all the scheduled changes are included in the release package and made available for systems testing. The entire release package is subjected to a physical audit by software SCM and validated by software quality assurance. The audit verifies and validates that all the items (e.g., document updates, test plans/procedures, version descriptions, etc.) are complete. After the successful completion of the audit, the release package is presented to a review board for approval to proceed with systems testing. This approval process helps reduce wasting system resources for testing when a product is not ready, and provides a substantial cost/schedule benefit.

SCM is responsible for updating the CSA database with the results of the review board decisions. Updated status reports are produced for management and the maintenance team. The results and reports generated by the audit are archived and become a permanent part of the release package documentation.

A.11.5 System testing

During the system testing phase, SCM is responsible for ensuring the integrity of

a) Test-case data
b) The software product on suitable media
c) Other test material

SCM provides the test group with up-to-date test material as requested. In test environments that are automated, SCM is responsible for maintaining version control over the test material (e.g., test-case drivers, regression test-case data, etc.).

When system testing is complete, SCM is responsible for archiving test material for the test group. SCM adds the test-report data to the archived release package documentation. Problems encountered during testing are documented and entered into the CSA database by SCM.

A.11.6 Acceptance testing

SCM provides total control of all the material made available to support acceptance testing. This material is returned to SCM at the completion of acceptance testing. SCM adds the acceptance test-report data to the archived release package documentation. Problems encountered during testing are documented and entered into the CSA database by SCM.

The review board is presented with all the test results, along with recommendations from each group in the maintenance team to allow an informed decision on the suitability of the system for delivery to the user community. SCM updates the CSA database with the results of the review board decision and provides management with current status reports.

A.11.7 Delivery

After approval by the review board and concurrence by project management, SCM is responsible for delivery of the system to the user community. Based on how the users access the system, the delivery may entail replacing the existing system with the new version, duplication from a master for delivery to remote users, or digital transmission to the users. Irrespective of the method, SCM is responsible for preparing and disseminating the system.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

In addition to the physical delivery of the system, SCM is responsible for updating the configuration index records to reflect the new version and archiving the complete system, including all release package data. SCM should provide copies of the entire system for disaster recovery storage.

A.12 Metrics/measures

Establishing and implementing a metrics plan is critical for providing insight regarding an organization's level of productivity as well as the quality of the software maintained by that organization. Additional guidance in the form of definitions, methodologies and rationale for implementing a metrics plan is provided in IEEE Stds 982.1-1988 and 982.2-1988. Metrics/measures captured for maintenance should enable the manager to manage the process and the implementor to implement the process.

To initialize a metrics process, management needs to identify technical factors that reflect the technical quality as well as management quality (i.e., effective use of schedule and resources) of the software being maintained. Once these factors are identified as indicators, then measures should be developed that correspond to the technical factors and quantify those technical factors. It is suggested that the selection of the technical factors and their corresponding metrics/measures be optimized such that only the factors that are most pertinent to the specific phase of the maintenance process are addressed during that respective phase of the maintenance process.

Once these are identified, a cost-benefit analysis should be performed to determine the best value that can be achieved by the organization (in terms of increased productivity and overall better management of the process) in exchange for the effort expended to collect and analyze the metrics/measurement data. At the very minimum, effort in terms of work hours should be collected and converted to cost using the organization's internal labor rate. Additionally, some measure of functionality as well as the error rates generated and classified by priority and type should be collected.

Some common measures of functionality are source lines of code (SLOC), function points, and feature points. Whatever method is chosen, it should be well-defined and agreed upon within the organization. Tools used for metrics/measurement collection, analysis, and assessment should be validated, calibrated, and used consistently throughout the organization.

In the world of software maintenance there are three major cost drivers that dominate the effort expended during the software maintenance process. These are documentation, communication and coordination, and testing. Therefore, any software maintenance metrics plan should include metrics/measures that accurately track performance based on these cost drivers, such as documentation change pages, efforts to negotiate the scope of the work to be included in the change package, and classification of the error rates by priority and type. Furthermore, any measure of functionality used should be tied to the complexity of the software as well as the application type (i.e., enhancement, repair, etc.).

A complexity profile of each program may be comprised of but not limited to the following:

a) Size of program (number of statements or instructions)
b) Number of modules in programs
c) Number of variables in programs
d) Number of global variables in programs
e) Average module size (in statements)
f) Average number of compares per module
g) Average number of modules accessing a global variable
h) List of common modules
i) List of modules that access more than the average number of global variables
j) List of modules that exceed the module size limit of 50 statements or exceed the module compare limit of 10 compares

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

The primary data source for the initial metrics/measurement input to the application profile is the software configuration library. Once the initial data are captured, an on-going maintenance metrics repository should be established. This repository should be directly interfaced with the organization's modification control system. The modification control system is a primary data source for the continuing update of the applications inventory profiles.

A.13 Software replacement policy

Planning for maintenance should be considered for all systems, even those under initial development. Although all systems eventually need maintenance, there comes a time when maintenance to an existing system is not technically or fiscally possible. Trade-offs as to resources, funds, priorities, etc., may dictate that a system should be replaced rather than changed. The management policy that can help in making a correct decision includes determination of the following:

a) System outages or failure rate
b) Code > *n* years old
c) Complex system structure or logic
d) New hardware
e) Excessive resource requirements
f) Missing or deficient documentation or design specifications.

Additional information can be found in NBS Publication 500-106, Guidance on Software Maintenance.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Annex B
Supporting Maintenance Technology

(informative)

B.1 Definitions

The definitions listed below define terms as used in this annex.

B.1.1 adaptive maintenance: Reactive modification of a software product performed after delivery to make a computer program usable in a changed environment.

B.1.2 emergency maintenance: Unscheduled corrective maintenance performed to keep a system operational.

B.1.3 formal unit: In reverse engineering, a unit of a system identified only by its links with other units.

B.1.4 functional unit: In reverse engineering, a unit of a system defined by its function; a functional unit may include one or several formal units, or be a part of a formal unit.

B.1.5 jump: Transfer of control.

B.1.6 re-engineering: A system-changing activity that results in creating a new system that either retains or does not retain the individuality of the initial system.

B.1.7 restructuring: The translation of an unstructured program into a functionally and semantically equivalent structured program. Restructuring transforms unstructured code into structured code to make the code more understandable, and thus more maintainable.

B.1.8 schematic: In reverse engineering, a description of links between system units; links are represented by jumps.

B.2 Re-engineering

Software maintenance is a significant part of the software engineering process. New code inevitably requires change: new regulations, changes to functions or the rules that compose functions, corrections to problems, extensions to functions, etc. These changes are typically treated as a minor part of the development process and delegated to "less experienced" programmers. It was generally assumed that these newly developed systems would have a short life span and be rebuilt once the mass of changes needed by the system became too costly to undertake. However, these systems have continued to be of significant value to the organization, and hence, the revitalization of these aging systems has become a practical option. As a subset of software maintenance, re-engineering has received a significant amount of recent attention. Redevelopment of key systems has become extremely costly in both dollars and disruption. A critical analysis of the software portfolio and selective re-engineering is a more evolutionary way of bringing old systems up to current standards and supporting newer technologies. A sound approach to re-engineering can not only revitalize a system, but also provide reusable material for future systems and form the functional framework for an object-oriented environment. The techniques to do this have always been available within other engineering disciplines. However, their application to software is a recent trend, and consequently, the tools to support software re-engineering are just now emerging.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Re-engineering as an approach is generally composed of two components: reverse engineering, and forward engineering. Reverse engineering does not change the system. It provides an alternate view of the system at a different level of abstraction. This generally means *redocumenting* code as schematics, structure charts, or flow diagrams to assist in understanding the logic of the code. Additionally, the process offers opportunities for measurement, problem identification, and the formulation of corrective procedures. Forward engineering is the process of system-building. This process begins with an existing system structure that is the framework for changes and enhancements.

Toolsets to support re-engineering are available and are evolving along the lines of CASE tools; single-function tools have begun to evolve to component toolsets that will evolve to integrated sets. These computer assisted re-engineering (CARE) environments will provide seamless reverse and forward engineering tools that are repository-based. In addition, measurement will play an increasingly important role in problem identification and resolution.

B.3 Reverse engineering

Many systems have the source code as the only reliable representation. This is true for a large long-lived system that has undergone many changes during its lifetime. These systems have substantially overgrown the initial base system, and have poorly updated documentation.

This is where reverse engineering is a recommended technique for redocumenting the system.

The first document that is needed to easily navigate in a system and to find the location of a problem at the analysis stage of system maintenance is a program schematic. A program schematic is the analog of an electrical schematic, which is the first document required for maintaining an electrical system.

Parallel definitions of schematic documentation are cited in table B1.

The process of reverse engineering evolves through the following six steps:

“In-the-small”—for local analysis on a unit level

a) Dissection of source code into formal units
b) Semantic description of formal units and declaration of functional units
c) Creation of input/output schematics of units

“In-the-large”—for global analysis on a system level

d) Declaration and semantic description of linear circuits
e) Declaration and semantic description of system applications
f) Creation of anatomy of the system

Steps and products of reverse-engineering a computer program are cited in table B2.

B.4 Holistic reusing

A functioning reliable system during its lifetime can give birth to a new, stand-alone system that has its own individuality. The functioning system then becomes a “parent” system. The new-born system is as reliable as the parent system (at least at the time of birth). The parent and the offspring systems concurrently exist and evolve. The process of “delivery” of a new system from the parent one is called *holistic reusing*. This differs from system maintenance activity, which occurs within the bounds of one system.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Holistic reusing is also a powerful tool for maintaining a system that can not be interrupted except for quick corrective emergency maintenance (to take the bugs out) while the work load is low. Perfective or adaptive maintenance may need a lot of system time that may not be available in the real world of a running business.

B.5 Software tools

Typical methods and tools that can be used during the maintenance process are listed in table B3. Information on the tools listed can be found in technical and vendor literature.

Table B1—Parallel definitions for schematic documentation

Electrical engineering	Software engineering
electrical schematic. A description of links between units of a device (links are represented by wires).	**program schematic.** A description of links between units of a program (links arerepresented by “jumps;” i.e., transfers of control).
Schematics “in-the-small”—for local analysis on a unit level	
I/O electrical schematic. A description of all wires connecting an individual electrical unit to other units.	**I/O program schematic.** A description of all transfers of control to and from an individual program unit.
Schematics “in-the-large”—for global analysis on a system level	
linear electrical circuit. A succession of consecutively connected electrical units.	**linear program circuit.** A succession of consecutively connected program units.
electrical application. A family of linear circuits executing a specific task of an electrical device.	**program application.** A family of linear circuits executing a specific task of a program.
electrical system anatomy. A list of all applications with relevant electrical circuits.	**program system anatomy.** A list of all applications with relevant program circuits.

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Table B2—Process and products of reverse engineering program schematics

Process steps	Products
1. Formally describe all links (transfers of control) in the program.	Formal units: **unit entrance.** A statement to which control is transferred from another unit, or the first program line. **unit output (exit).** A statement from which control is transferred to another unit. **unit end.** A statement preceding the entrance of another unit. **subroutine unit.** A unit that ends with a RETURN-like command. **non-subroutine units:** **transiting.** A one-to-one unit. **branching.** A one-to-many unit. **rooting.** A many-to-one unit. **starting.** A none-to-one unit. **ending.** A one-to-none unit. **unit number.** A number assigned to a unit from two sets of consecutive numbers—one set for subroutine units, another for non-subroutine units.
2. Semantically describe function for each formal unit, and create functional units.	A functional unit consisting of one or several formal units or being a part of a formal unit.
3. Describe links for each unit.	An input/output (I/O) schematic that is combined with a segment of code belonging to a unit.
4. Create a map of all units.	Program linear circuit (LC) **FIRST unit.** A unit from which LC starts. **LAST unit.** A unit at which LC ends.
5. Create a map of all FIRST and LAST units.	Program applications
6. Create a list of all applications with relevant linear circuits.	Program anatomy

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.

Table B3—Methods and tools for maintenance

Activities/methods	Tools	Manual	Auto
I. Problem/modification identification			
1. Modification/problem reproduction	Automatic test equipment		X
II. Analysis			
1. Code beautifying	Beautifiers		X
2. Reverse engineering program schematic			
a. “in the small”			
• Declaring formal units	Diagrammer	X	X
• Declaring functional units via dissecting/combining formal units	Expert system	X	X
• Mapping input/output schematic for each functional unit	Mapper		X
b. “in the large”			
• Mapping functional units, and declaring program linear circuits	Mapper	X	X
• Mapping program linear circuits, and declaring program applications (families of linear circuits)	Mapper	X	X
• Creating anatomy of program		X	
• System metrics/measures	Metric analyzer	X	X
3. Code restructuring	Structure analyzer	X	X
III. Design			
1. Reverse engineering design documentation			
a. Flow-charting	Diagrammer	X	X
b. Data-flow diagramming	Diagrammer	X	X
2. Visualizing	Visualizers	X	X
3. Documenting changes (large)	Documenter	X	X
4. Documenting changes (small)	Documenter	X	X
IV. Implementation			
1. Code generation	Code generator		X
2. Code analyzing	Code analyzer	X	X
3. Simulation/emulation	Simulators & emulators	X	X
4. Test analyzing	Test analyzers	X	X
5. Test data generation	Test data generators		X
6. Profiling	Profilers	X	X
VI. System/acceptance testing			
1. Stress testing		X	X
2. Performance testing	Performance monitors	X	X
3. Function testing		X	X
VII. Delivery			
1. Media duplication/verification	Media duplicators/verifiers	X	X

Authorized licensed use limited to: DUKE UNIVERSITY. Downloaded on May 04,2012 at 19:46:02 UTC from IEEE Xplore. Restrictions apply.