

IEEE Recommended Practice for Information Technology— Year 2000 Test Methods

Sponsor

**Portable Applications Standards Committee
of the
IEEE Computer Society**

Approved 26 June 1999

IEEE-SA Standards Board

Abstract: This document provides users of computer hardware, firmware, software, or data systems with recommended practices for assessing and demonstrating the system elements within their organization that may be at risk of failure due to the Year 2000 problem and related date-specific issues. This recommended practice provides the framework for detailed planning and execution of all steps and tasks involved in testing for Year 2000 compliance. The resulting plan will outline the testing approach and identify system elements that are at risk of failure when crossing into the Year 2000 or using data that includes dates after 2000-01-01.

Keywords: bridge, date, real-time clock (RTC), remediation, rollover, system time, test, Year 2000, Y2K

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

Copyright © 1999 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 30 December 1999. Printed in the United States of America.

Print: ISBN 0-7381-1779-X SH94773
PDF: ISBN 0-7381-1780-3 SS94773

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

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. 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 societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

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

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

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

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

Introduction

(This introduction is not part of IEEE Std 2000.2-1999, IEEE Recommended Practice for Information Technology—Year 2000 Test Methods.)

This recommended practice provides the framework for detailed planning and execution of all steps and tasks involved in testing for Year 2000 compliance. The resulting plan will outline the testing approach and identify system elements that are at risk of failure when crossing into the Year 2000 or using data that includes dates after 2000-01-01. The focus of this recommended practice is to ease the transition for companies through Year 2000 by outlining a range of example test cases to aid and expand the capabilities of the individual companies to assess, test, and verify software, hardware, and firmware. This recommended practice should be used with IEEE Std 2000.1-1998, IEEE Standard for Year 2000 Terminology.

This recommended practice describes the process, test methods, and remediation for Year 2000 assessment and validation. It is intended for individuals or organizations that develop, test, acquire, or use software, firmware, or hardware.

This recommended practice is designed to help individuals and organizations. It does not attempt to develop a comprehensive test suite. It is expected that systems affected by Year 2000 related problems will range from single chip embedded systems to global networks. Since test procedures are typically system-dependent, it is impossible to address every Year 2000 problem. Instead, included are examples of sample test cases that are intended to spur investigation into similar functions in a specific application. These sample test cases are intended to be generally applicable. Furthermore, the test cases are intended to be problem-specific and not application-specific and may have applicability to a wide variety of applications or situations. For this reason, application-specific terminology has been avoided where possible. It is likely that a specific system may need hundreds of test cases related to a single type of module. The process of writing the test case steps and results will be facilitated using the sample test cases. It should be noted that this recommended practice focuses only on the century digit change problem, it does not address date counter overflow.

WARNING: Testing for Year 2000 related issues on operational systems might cause damage to the system or the data contained therein. Some of these issues will be further addressed in this document.

Organization of this recommended practice

The standard is divided into 13 clauses

- Overview: Statement of scope and conformance information (Clause 1)
- References: List of normative references (Clause 2)
- Definitions: Terms as used in this document (Clause 3)
- Editorial conventions and warnings (Clause 4)
- Methodology: Creation of a test plan (Clause 5)
- System elements at risk: Example test cases (Clause 6)
- Informative Annex A through Annex G

IEEE Std 2000.2-1999, IEEE Recommended Practice for Information Technology—Year 2000 Test Methods, was prepared by the P2000.2 Working Group of the IEEE Computer Society. At the time this recommended practice was approved, the membership of the IEEE PASC working group was as follows:

Participants

PASC Sponsor Executive Committee

Chair: Lowell G. Johnson
Vice-Chair: Joseph Gwinn
Functional Chairs: Andrew Josey
Jay Ashford
Curtis Royster
Secretary: Nick Stoughton

At the time IEEE completed this document, the Year 2000 Test Methodology Working Group had the following membership:

Lowell Johnson, Chair
Christina Drukala, Technical Editor

Alan Belcher
Steven Brock
Johnathon Chapman
Eldon Colby
John S. Davies
Nelson deGrandmaison

David Dodd
Don Estes
Vincent E. Henley
Victor Grebler
Mary Hengstebeck
Thomas P. Koenig
Kevin Lewis

Bob Lynch
John Napier
Alan Peltzman
John Russell
Terrill J. Slocum
Gary R. Young

The following members of the balloting committee voted on this standard:

William B. Adams
Keith Andrews
Jay Ashford
Ida M. Berchtold
Adam Block
P. Botman
Mark Brown
Donald Cragun
Ming De Zhou
Christina Drukala
Philip H. Enslow
William Eventoff
Michel P. Gien
L. M. Gunther
Bryan D. Harold
Barry Hedquist
Vincent E. Henley
Thomas R. Hunwick

Jim D. Isaak
Richard A. Jaenicke
Petr Janecek
Jerry L. Johnson
Lowell G. Johnson
Andrew Josey
Leon A. Kappelman
Judith S. Kerner
James J. Keys
Thomas P. Koenig
Kenneth C. Kung
Thomas M. Kurihara
Steli Loznen
Austin J. Maher
Robert A. Martin
Roger J. Martin
James W. Moore
Finnbarr P. Murphy

John Napier
Paul D. Nugent
Marc A. Pearl
Alan N. Peltzman
C. J. Riddick
Curtis Royster
Christoph Ruland
Alan Russell
Michael D. Shapiro
Terrill J. Slocum
William R. Smith
Sandra Swearingen
Efsthathios D. Sykas
Michael W. Vannier
Dale W Way
Paul A. T. Wolfgang
X/OPEN CO. LTD.
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 26 June 1999, it had the following membership:

Richard J. Holleman, *Chair*
Donald N. Heirman, *Vice Chair*
Judith Gorman, *Secretary*

Satish K. Aggarwal
Dennis Bodson
Mark D. Bowman
James T. Carlo
Gary R. Engmann
Harold E. Epstein
Jay Forster*
Ruben D. Garzon

James H. Gurney
Lowell G. Johnson
Robert J. Kennelly
E. G. "Al" Kiener
Joseph L. Koepfinger*
L. Bruce McClung
Daleep C. Mohla
Robert F. Munzner

Louis-François Pau
Ronald C. Petersen
Gerald H. Peterson
John B. Posey
Gary S. Robinson
Akio Tojo
Hans E. Weinrich
Donald W. Zipse

*Member Emeritus

Also included is the following nonvoting IEEE-SA Standards Board liaison:

Robert E. Hebner

Noelle D. Humenick
IEEE Standards Project Editor

Contents

1.	Overview.....	1
1.1	Scope.....	2
1.2	Purpose.....	2
1.3	Assumptions.....	2
2.	References.....	2
3.	Definitions.....	3
4.	Editorial conventions and warnings.....	4
4.1	Editorial conventions	4
4.2	Warnings	4
5.	Methodology	6
5.1	Year 2000 life cycle	6
5.2	Test process.....	6
5.3	Impact of external constraints on Year 2000 testing strategy.....	10
5.4	Test impact of remediation techniques	12
5.5	Special considerations involving embedded systems	14
5.6	Special-condition dates	18
5.7	Special date conditions	22
5.8	System components to be tested	25
6.	System elements at risk.....	26
6.1	Interfaces: Shared control blocks/API/DDE/OLE.....	26
6.2	Archiving/restoring.....	27
6.3	Backup and restore	33
6.4	Calculations	36
6.5	Date determination	40
6.6	Hardware	44
6.7	Computer numerical controls	46
6.8	Communication protocols.....	47
6.9	Compilers.....	48
6.10	Event-triggers	52
6.11	Error handling.....	56
6.12	File access system.....	58
6.13	Globalization/internationalization	61
6.14	Synchronization of distributed networks.....	62
6.15	Import/export	66
6.16	Multisystem windowing	70
6.17	Licensing	71
6.18	Logs/date stamps	73
6.19	Merge.....	75
6.20	Parsing/validation	78
6.21	Performance.....	81
6.22	Operational time periods	82
6.23	Queries, filters, and data views.....	84

6.24	Data recovery	85
6.25	Bridge testing.....	87
6.26	Sorting	90
6.27	User interface (input and output).....	93
6.28	Date format	94
Annex A (informative) Search strings		96
Annex B (informative) Dates		97
B.1	Leap years.....	97
B.2	Flags 01-01-99 – 12-31-00	97
B.3	Year 2000 rollover 12-31-1999 – 01-01-2000	97
B.4	Counter rollover	97
Annex C (informative) Example archive documentation		98
C.1	Overall organization and responsibilities for Year 2000 compliance	98
C.2	Specification of system under test.....	99
C.3	Test environment specification	99
C.4	Summary of general test methods and strategies	99
C.5	List of areas tested.....	99
C.6	List of areas not tested.....	99
C.7	Copy of all list cases executed against individual builds	99
C.8	Upper-bound/lower-bound dates	99
C.9	Exception logic/error handling.....	99
C.10	Input/output	100
C.11	Remediations	100
C.12	Compatibility.....	100
Annex D (informative) Alternative testing methodology.....		101
D.1	Alternatives to comprehensive testing.	101
Annex E (informative) Coverage overview.....		104
E.1	Level of testing	104
E.2	Coverage analysis	104
E.3	Functional analysis	105
Annex F (informative) Date systems		106
F.1	Gregorian calendar details.....	106
F.2	Julian calendar.....	106
F.3	Julian day calendar	106
F.4	Lilian day calendar	106
F.5	Modified Julian day number	107
Annex G (informative) Bibliography.....		108

IEEE Recommended Practice for Information Technology— Year 2000 Test Methods

1. Overview

This recommended practice is designed to assist organizations and individuals in both the assessment of their assets for Year 2000 status and validation of their Year 2000 readiness after assets have been remediated. The problem domain that the technology community faces is how to assess system elements within their organization that may be at risk of failure due to Year 2000 related problems. This problem may affect software, firmware, hardware, and data elements. Private enterprise, government agencies, and the military all face the possibility of failures. Because the problem is pervasive, this recommended practice will not focus on any particular platform, technology, or industry. This recommended practice is not a certification. It is a template that can be customized by each organization to help in the creation of Year 2000 Test Plans and Test Scenarios for the technical assessment, verification, and validation of their individual assets. Each organization will need to prioritize and apply good engineering test methodology when executing Year 2000 testing activities.

The procedures in this document are not intended as a certification. This document's intent is to serve as an outline to help guide personnel responsible for testing Year 2000 problems to isolate troubled areas found during Year 2000 remediation. Subsequent clauses provide a nonexhaustive list of system elements and components that may have difficulty transitioning into the Year 2000. The list should be compared against the functions and procedures used by the system; similar system elements should be tested. The system elements at risk and the sample test assertions are to be used cooperatively to enable a testing organization to create customized test assertions for specific assets, whether software, firmware, hardware, or data elements.

The body of this recommended practice is divided into six areas. Clause 1 contains information on the scope and purpose of the document. Clause 2 lists documents necessary for implementation of the procedures in this document. Clause 3 contains definitions of terms used within this document. Clause 4 gives a description of the test case outline used in Clause 5 and provides important warnings. Clause 5 supplies guidance on methodology, process, and the testing impact of different remediations, validation, and reporting for testing of Year 2000. In Clause 6 there are example system elements and features that have a high probability of being affected by date-based algorithms that could fail if not designed to operate on both sides of the 1999 to Year 2000 boundary. This clause also contains example test assertions that are generic in nature and correlate with system elements. The final clauses are annex data that may be valuable, and which are provided for informational purposes only.

1.1 Scope

This document identifies recommended practices for defining a user-specific test plan for Year 2000 validation. Included are:

- a) A taxonomy of system elements and features that are likely to exhibit Year 2000 failures;
- b) Test scenarios that may be used as templates for the creation of test cases to detect defective system elements with respect to the Year 2000; and
- c) A customizable methodology and process for determining the testing impact of various remediation techniques, validation techniques, and documentation of test results.

Significant engineering judgement must be exercised by the user of this document in applying its recommendations; it is not intended that these practices be employed in a mechanical or rote fashion. They should be used to supplement standard testing procedures that encourage proper coverage and consistency of testing.

1.2 Purpose

This document provides users of computer hardware, firmware, software, or data systems with recommended practices for assessing and demonstrating the system elements within their organization that may be at risk of failure due to the Year 2000 problem and related date-specific issues.

1.3 Assumptions

The examples and test cases used throughout this document assume the following conditions are true unless otherwise stated.

- Existing systems are complete, stable, and tested, and all features work according to a specification, documentation, or a functional baseline or a specific agreement among the parties.
- Once changes to the code have been made to correct Year 2000 problems, a regression test or appropriate function test is executed to validate that system functionality, including impacted date functions, are working correctly.
- The system element's date/time formats are convertible to the Gregorian Calendar for the relevant range of dates over which the test cases will run.
- Testing is performed in an environment that will not damage or impact real-world data. (See 4.2, Warnings.)
- The test facility has the ability to change the time clock of the system or simulate its change and, in some cases, age data.

2. References

The following standards contain provisions that, through references in this text, constitute provisions of this recommended practice. This recommended practice shall be used in conjunction with the following standards.

IEEE Std 2000.2-1999, IEEE Standard for Year 2000 Terminology.¹

¹IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (<http://www.standards.ieee.org/>).

ISO 8601: 1988, Data elements and interchange formats—Information interchange—Representation of dates and times.²

3. Definitions

3.1 acceptable deviation: In the context of evaluating specific test-case post conditions, a deviation is permitted based on an informed decision to specify that deviation as noncritical.

3.2 bridge: A bridge is a system element which converts from one data format to another. Bridging can be incorporated in most Year 2000 remediations to interpret date-data formats. This may be helpful in transferring dates between date formats for remediated system elements and those used in the original system. There may also be situations in which multiple remediation techniques requiring different date formats are used, creating a need for bridges between them.

3.3 failure: Any deviation from specified post condition of a test case is considered a failure for that specific test case. Post conditions should require adherence to the specification, documentation, or functional baseline for the system.

3.4 future regression: Regression testing provides a quick way to test broad areas of a system's functionality. Future regression testing expands the normal regression process to include future data and advanced system dates.

3.5 ordinal date (OD): As defined in ISO 8601: 1988.

3.6 pass: The lack of any deviation from the expected post condition of a test case signifies a pass for that specific test case. Adherence to specification or documentation, or functional baseline indicates a pass.

3.7 real-time clock (RTC): A hardware system element that provides the system with a reference to real-world time. A common implementation would be a circuit containing a set of registers holding the current month, day, year, day-of-week number, and other time-related values, along with a circuit that continuously updates these register values. The circuit is normally provided with an alternate power source, such as a battery, which allows the RTC to continue to function when main system power is not available.

3.8 special logic: Many programs use specific dates to trigger exceptions to normal date processing. A common example is expiration date on tape archives. Rather than adding another flag to the tape header, the date 1999-9-9 is used by many systems to mean never expire. Other dates have been used to indicate an unknown or out-of-range date. Since there is no standard for this practice, the specific dates used may be difficult to trace. **The potential for dramatic failures increases in 1999 due to date code flags and various other Year 2000 problems.**

3.9 system time: The state of any system element that is used to synchronize system events with real-world events based on a date, time, or combination of the two.

3.10 user-defined date systems: Some users may implement locally developed date systems to track user-specific events. For example, manufacturing, production, or just-in-time (JIT) dates. Methodologies developed for verification of Year 2000 issues associated with the Gregorian calendar may need to be adapted to apply to user-defined date systems.

3.11 Year 2000 rollover: The instant when a system's year changes from 1999 to 2000. In a system that uses a six-character date format this is a transition from 99-12-31 (YY-MM-DD) to 00-01-01 (YY-MM-DD).

²ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH-1211, Genève 20, Switzerland/Suisse (<http://www.iso.ch/>). ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).

4. Editorial conventions and warnings

4.1 Editorial conventions

4.1.1 Test case format

The sample test cases included in this recommended practice have been made as general as possible. Every effort has been made to not rely on the feature set of a specific system. In many situations the actual process necessary to execute a step may be much more complicated than indicated in these test cases.

Test cases are developed to satisfy various test objectives. Information elements common to a test or test case include test scenarios, test objectives, test conditions, test procedures, and expected test results.

4.1.1.1 Test scenarios

Most test cases are preceded by a paragraph explaining the choices made in the test design and their intended results. In many cases the test data or procedures will need to be modified in order to fit a specific situation. The initial paragraph is intended as an aid to understanding the purpose of the test case so that it can be used as a model for other test case designs.

4.1.1.2 Test objective

This is a one- or two-line statement of the purpose of the test case.

4.1.1.3 Test conditions

Settings that need to be modified from normal operating parameters are noted in this area.

4.1.1.4 Test procedures/expected test results

Procedures are actions necessary to cause an observable result. The test format used here presents the procedure, the result, and a step number in a table to enhance readability. In some cases several actions may be required to produce a single result. In these instances, the result column is left blank until the step where the final action in the sequence is completed.

4.1.1.5 Test dates—holidays

Holidays vary by jurisdiction in terms of both legal and cultural context. However, these may require specific action related to pay rates, intrusion alarms, or secure-area access. Examples of holidays are used in this document, mostly referencing U.S. national holidays. Users of this document should be aware of the diversity and variation of holidays where their operations are concerned, and adapt these examples for these contexts.

4.2 Warnings

Warnings are provided to highlight potential areas of risk resulting from testing. This subclause may provide suggestions on how to avoid loss or damage to system or system elements. Ensure that a disaster recovery plan is up to date and *operational* for restoration of production data in case of loss. Fully outlining a process for creating a disaster recovery plan is outside the scope of this document.

As noted in 5.2.4, Build test environment, it is best to conduct all tests on a duplicate system, when possible.

Complete backups of all corruptible data storage devices are essential. Consider maintaining a second set of media (hard disks on sliding brackets, disk packs, tapes, etc.) with the unremediated system installed on them. Embedded systems may pose special safety problems both because they often operate autonomously and because they may control systems with the potential to do physical harm to equipment, personnel, or to the public at large.

One way to alleviate these risks is to remove system-control elements from the larger system. For example, disconnecting them from physical outputs and connecting them to a device capable of displaying the output state of the controller. These devices can be as simple as a multimeter or breakout box. Systems with many control outputs or critical interactions between input sensors and outputs may require software simulations. A simulation can be programmed to respond to the system elements control outputs to simulate the timing or function of mechanical devices and actuate the elements sensor inputs.

While remediation of the system is sometimes done entirely in a software simulation, the verification and validation of the system elements should be done using a hardware interface that allows the actual hardware component to be plugged into the simulation.

Simulation can be used to detect the majority of errors before the embedded controller is tested in the full system. However, no simulation is perfect. Discrepancies between the simulation and the real system can cause failures. Careful real-world testing should be conducted after testing within the simulation is completed.

4.2.1 automated purge: Setting the system date to a future date could trigger purge routines causing data loss.

4.2.2 compiler preprocessors: Extra caution may be warranted for compiler preprocessors where the date retrieval mechanism may differ from and override that of the compiler.

4.2.3 data integrity: Ensure that data integrity is maintained during reintegration of system elements after testing and validation.

4.2.4 data loss: Adequate backups should be made prior to execution of tests. The potential for dramatic loss of data increases in 1999, due to date code flags and various other Year 2000 problems.

4.2.5 hardware damage: Use of future dates in system elements could damage hardware in embedded systems. For example, some system elements have maintenance triggers that force the system to fuse hardware components if not maintained within the periodic maintenance schedule causing physical damage to hardware.

4.2.6 security access: Ensure that security accesses are not lost after manipulating dates on system. Changing a system date to a future date may result in the loss of some or all access codes to the system. In some systems, once a date is advanced for test purposes, it becomes impossible to retract the date without a full system reinstallation.

4.2.7 software license expiration: Software with enforced licensing cannot be tested with the system clock set beyond the expiration. The software vendor should be contacted to arrange special licensing privileges.

4.2.8 software license violation: The creation of a testing environment by installing copies of an application could violate licensing agreements and copyrights. This may trigger electronic Licensing Managers to restrict active concurrent copies of the application.

4.2.9 system corruption: Systems whose dates are set to the future may suffer operating system corruption upon reset to present date. For example, system logs may be corrupted with invalid system time values. In cases like these, restoration or reinstallation of the operating system may be necessary.

4.2.10 systems integration: Incomplete or incompatible remediation techniques within a larger system can lead to system or data corruption, either during testing or during reintegration. For example, incompatible date formats can lead to corrupted data.

5. Methodology

This clause outlines the Year 2000 life cycle, and details the test methodology, as it applies to the Year 2000 problem, and the use of test methodologies for assessing and testing the systems elements at risk.

5.1 Year 2000 life cycle

A conversion model is comprised of five phases, each representing a major Year 2000 activity. Both the private and public sectors have used this model in addressing their respective Year 2000 issues. The five phases are described as follows:

- a) *Planning and awareness:* Define the Year 2000 problem and gain executive-level support and sponsorship for establishing the problem as a high priority item for resolution. Research and establish a project plan, and obtain budget and resources. Note that the planning activities are also relevant to the other phases described below.
- b) *Assessment (inventory):* Evaluate the Year 2000 impact on the enterprise; develop contingency plans to handle data exchange issues and system failures (dysfunction or system crashes); prioritize systems by identifying those that are mission-critical.
- c) *Remediation (renovation):* Convert, replace, eliminate, work around, or encapsulate one or more system elements; modify interfaces.
- d) *Validation (audit):* Test, certify, and validate all system elements that have been converted or replaced.
- e) *Implementation:* Place into production all system elements that have been converted or replaced.

This recommended practice does not address the subject of validation within the implementation phase, which is generally outside the scope of testing/validation organizations. However, due to potential interdependencies between any remediated/validated system and the operational environment within which it must function, there is a likelihood that additional problems will emerge in the implementation phase. The origins of these problems could be in the remediated/validated system, in some other system(s) in the environment, or in the interactions among them. Therefore, it is recommended that an effort be made to plan, audit, and manage the implementation phase with the goal of detecting, locating, and addressing such problems.

5.2 Test process

Experience with Year 2000 projects indicates that a major portion of the effort is taken up by testing. Limited resources, high cost, and restricted project duration make it essential that organizations and individuals use an efficient and effective testing process.

5.2.1 Establish a test strategy

It is key to success that the testing strategy be largely driven by business, not just technical considerations. It is important to ensure that there is no disconnect between those responsible for the general management of the business and those responsible for its information technology. Neither party in this partnership can arrive at the optimized testing strategy for the enterprise alone. Technical managers have a responsibility to ensure that business managers have the most current and accurate information on which to base key decisions. Busi-

ness managers have a responsibility to understand the risks described in technical terms and to ensure that appropriate funding and resource decisions are made to achieve the best possible outcome for the enterprise.

A testing strategy is top-level guidance concerning the nature, constraints, and desired outcomes of a testing effort. It guides the development of all the specific test activities that follow. Elements of a strategy can result from testing impacts of the type of system or system elements being tested. Factors critical to the success of the testing effort are identified. Level of risk and appropriate level of testing effort are identified. The test scope and who will perform testing is determined. Trade-offs are identified between competing planning factors such as schedule, cost, scope, technological approaches, and quality.

Many organizations may find that there are not enough resources or time to test all their applications for Year 2000 compliance in a conventional or comprehensive manner. For those organizations, a prioritization of business functions and their supporting systems should be determined. The strategic priority is based on the risk to the enterprise should systems fail. This prioritization results in a series of risk levels being defined that can be applied to an organization's systems. These risk levels determine the level of testing effort required for each system. An example of such a risk hierarchy is presented in Annex D.

It is appropriate to understand from the outset what the available time, human, and machine resource constraints are. The test strategy and subsequent test plans must fit within the envelope of what can be done. It is inappropriate and futile to create plans and strategies that cannot be executed within the available constraints. It is strongly recommended that determining this envelope of constraints be the first order of business before elaborate, but unusable plans are developed.

The level of testing effort and criteria for completion for a system depends on the strategy, risk assessment, and remediation technique selected. Completion criteria are often expressed as the degree of functional and/or code coverage achieved by the testing effort. Testing effort can range from the decision not to test through exhaustive testing. Testing effort can have a direct correlation to the risk of operational safety.

Year 2000 testing also requires that the differences between system times and asynchronous data times are understood and that appropriate date intervals are used that are sufficient and appropriate for the system being tested. System time is determined by clock mechanisms on the system being tested. The system clock provides the increment of time that synchronizes time date-data processing information for both function-specific purposes and system operational requirements such as internal logs or file time stamping. Data time has to do with the dates used within the data being processed, which can be current data or data aged beyond 2000 to reflect a future functional need. Testing of both system time and data time is necessary, but the dates selected should be appropriate for the system under test. For example, the valid date interval for a strategic planning system could be from the present to the year 2030. The valid date interval for an income tax computation system could be a year or less.

5.2.2 Set test objectives

Test objectives are an identified set of system elements to be measured under specified conditions by comparing date-related behavior with the expected behavior. The process of establishing Year 2000 test objectives may include a review of the test strategy and system requirements. Considering such factors as size, complexity, environment, and reliability can be helpful in assessing risk associated with the system elements. Testing objectives should be defined to address portions of system elements that have high risk and completion criteria. The test completion criteria must be clearly defined and measurable.

5.2.3 Develop a test plan

The test plan is a document describing the scope, approach, resources, and schedule of intended test activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning. The test plan should clearly specify the metric that defines the end-point of the test so that all concerned parties can agree when the test is complete. There are many different types of

metrics, for example, the number of test cases run successfully, defect discovery rate threshold, or mean time to failure.

The test plan should be appropriate to the test strategy, objectives, and risk. Examples of levels of risk and the associated levels of testing effort are presented in Annex D.

5.2.3.1 Define tests

The full range of system functions that depend on the remediated system elements are tested to insure that changes do not adversely affect related functions.

Tests address the following considerations in order to satisfy the test objectives:

- **Functional attributes.** Determine the functionality of the system elements impacted by date variables. Determine the expected outputs from the operational logic that acts on these variables. This can be approached from a data-oriented input versus output perspective (black box) or from a logic perspective (white box). In order to meet the testing criteria established for the system, it may be necessary to analyze data structures, code logic, or the use of reserved dates or special dates (e.g., quarter end, month end) in the code or operational logic.

The use of aged data is a necessary, but not sufficient, condition for successful Year 2000 testing. Both data, which are aged statically, that is, converted permanently to an aged condition, and dynamically, converted in process, may be needed. Changing the system clock does not age the data.

Aged data are data that have been modified such that date variables within that data are changed from their current value to some future date. Normally, the year value must be changed to be useful in a Year 2000 testing context. Care must be taken to understand such things affected by the change of year, such as day of the week and leap year, and to either ensure consistency or to account for that difference in any testing scenario. An alternative to modifying existing data to an aged state is to create test data that has the same aged characteristics as modified existing or production data. This data may be data that has no other reason for existing other than Year 2000 testing.

- **Nonfunctional attributes.** Evaluate the structural attributes of the system under test to identify those attributes that may require date-related testing that are not dependent on the system functionality. Such date-related factors include performance, usability, maintainability, reliability, availability, serviceability, portability, extendibility, and security. For example, the use of complex windowing remediation techniques may reduce the readability of the program code. Portability may be a concern if program code is changed on one platform and compiled or executed on another.
- **System time, data time, and valid date intervals.** As noted in the discussion of strategy, testing of both system and asynchronous data time is necessary, but the dates selected should be appropriate for the system under test.
- **Remediation technique impacts.** Testing impacts of various Year 2000 remediation techniques are described in 5.4.
- **Special-condition dates.** Consider the special-condition dates in 5.7.
- **Existing test cases.** Evaluate existing tests and test data. Some of the needed tests may have already been developed during normal development testing or the assessment phase.
- **System elements at risk.** Review the sample test cases to identify system elements at-risk and to assist with the creation of tests appropriate to the system. See Clause 6.

5.2.3.2 Develop test cases

Once tests have been defined, test cases should be developed to fully exercise the requirements of the test. Information found within a test case typically includes test objective, test conditions, test procedures, and

expected test results. As test cases are developed, other components necessary to perform the tests should be defined. For example, test sequencing, test procedures, testing tools, and sources of test data are identified. Tool requirements can include file comparison tools, date simulation tools, date-data aging tools, transaction capture tools, and tools for logical test path analysis.

5.2.4 Build test environment

The next step of the process is to create the necessary environment to support testing, for example, develop test simulations, test automation, and test data.

Creating the test environment requires care because of the dangers listed in 4.2.

Critical systems that must remain in operation during remediation and testing may require that an isolated hardware environment be used. A baseline should be created for this system to ensure that the unremediated test system accurately represents the original system. Hardware cost and availability may make exact duplication of the original system unfeasible. It may be possible to perform initial tests and remediate on some systems using scaled down hardware. It should be noted that any difference between the original system and the test environment introduces a risk that the remediation will not be successful when moved to the original system. Care should be taken to minimize this risk as much as possible. It is highly recommended that the final remediation of the code be tested as thoroughly as possible on a hardware environment equivalent to the system it is to be implemented on.

As test cases are being developed, the need for test data to satisfy test case condition requirements will become clear. Test data can come from existing test cases or files. Data to support specific Year 2000 test cases can be created. Production data can be copied and used, but will not contain many high-risk dates or conditions. Adding specially developed test data to production data in order to satisfy high-risk date testing requirements may help to overcome this shortcoming. Consider the use of date simulators to evaluate artificially changed calendar dates and the use of an appropriate tool to age data.

Create test data of dates on both sides of the end points of the valid date intervals, not just of the end points and the range within them. Testing invalid dates will provide an indication of the system's response in such cases. Make sure all relevant "special dates" are in test data and dates on either side of them, as well.

5.2.5 Test execution

Test execution is the phase most likely to result in damage or loss to the system being tested.

As tests are being executed, special care should be taken to accurately identify and record those anomalies related to Year 2000.

The remediation technique may influence the selection of test tools to evaluate test results. For example, encoding may require a tool to interpret the codes used and to report the results.

Before executing a test case and between test case executions, it is necessary to assure the system is in the test base state. Restoring the system to the test base state after a test run may involve a number of actions including, in no specific order

- Restoring the system time clocks, calendars;
- Reconnecting any network or system;
- Removing any test files and restoring the system to pretest conditions;
- Re initializing (rebooting) the equipment; and
- Restoring the test data.

5.2.6 Test verification and validation

5.2.6.1 Verification

Verification requires that a correlation be achieved between the expected and observed results of a test case. Documentation should record either a correspondence between the two or make note of deviations. If failures occur that prohibit the completion of tests, the test cases should be documented along with the circumstances that precluded their use.

5.2.6.2 Validation

5.2.6.2.1 Test plan validation

Validation of the test plan attempts to ensure that the testing process is comprehensive.

As an example, statistics gathered by tools that evaluate the number of branches or statements executed during testing versus the total number in the system's code are useful in showing the completeness of regression testing.

The techniques in E.2, Coverage analysis, are useful in creating a valid test plan. If these methods or similar techniques were used during test planning, their application should be documented during validation.

5.2.6.2.2 System validation

The validation process determines if the system is able to perform the function it was intended to perform. For the purposes of this document, it is assumed that the system under test has been validated at some time. During the assessment and remediation process some functionality of the system may be abandoned in order to reduce the length of the Year 2000 process. System validation requires that checks be made to ensure that any changes to functionality have not affected the systems ability to perform its primary functions.

5.2.7 Summarizing and reporting

After the test cases that have been planned are executed, the test report is produced. Findings and recommendations are determined. Testing is complete when test completion criteria are met. Typical measures include tracking test cases completed to test objectives and/or measuring other coverage criteria.

Due to the potentially high business and safety risks associated with Year 2000 failures, it is important that test documentation be archived. See Annex C.

5.2.8 Test reuse

In order to facilitate reuse of test cases, procedures, and materials, they should be archived in such a manner as to recreate the test environment. The need to periodically recreate the test environment should be recognized and planned for deliberately. Year 2000 related testing is not a one-time event. Routine changes in system functionality will continue to occur and will need to be tested. Regression (baseline) testing of all functions whenever change occurs is necessary due to the unanticipated effects of changes in one area upon other areas.

5.3 Impact of external constraints on Year 2000 testing strategy

Technological constraints, business objectives, and the public welfare may be considered in designing an appropriate test strategy. The integration of a well-defined Year 2000 testing methodology will increase the likelihood of Year 2000 success as defined by the business goals of the project. Each system-in-test will have

individual system elements that are at risk. Understanding the consequences of failure of the components can aid in prioritizing the focus of resources on individual testing tasks. Decisions as to whether any specific test methodology should be employed should be made against the background of these more general considerations. There are no universal solutions for the Year 2000 test problem. Factors to be balanced in deriving a test strategy may include, but are not limited to, the following business/public considerations.

5.3.1 Business/public considerations

- Lost business
- Cost of failure
- Cost of litigation
- Regulatory impact
- Public impact
- Public safety
- Shareholder value of public companies
- Fiduciary responsibilities of corporate or organizational officers

5.3.2 Likelihood of failure

- Amount of changes
- Type of remediation
- Complexity of system elements
- Possibility of workarounds
- Interoperability of systems (i.e., contagion of failure)

5.3.3 Parallel development

Introducing non-Year 2000 related improvements along with Year 2000 remediation jeopardizes Year 2000 testing. This can invalidate the current Year 2000 test baseline, making Year 2000 test results difficult to interpret.

5.3.3.1 Restrict new features

New features can introduce new requirements for system elements that have already been validated in the Year 2000 test process. This may require retesting all or part of the elements previously validated.

5.3.3.2 Isolate elements

It may be necessary to temporarily isolate elements of a system. The isolated section of the system should be restricted from upgrades and bug fixes. Year 2000 remediation and testing can then be performed on the isolated component, while other elements remain available for normal development. As elements are validated, they are integrated into the development system and restrictions are lifted. Then a new area is isolated and restricted, and the process repeats. Since development is restricted to areas not currently being remediated in the Year 2000 process, they can be integrated more easily into the Year 2000 system, thus limiting divergence of the systems.

5.3.3.3 Synchronizing remediation with ongoing development

The pervasive nature of some Year 2000 remediations can mean major structural system changes that require months or even years to complete and test. During such an extended period of time, the system being revised for normal development may diverge so much from the Year 2000 remediated code that combining and retesting becomes nearly impossible. Breaking the remediation and testing process into discrete components that can be completed quickly allows systems to be synchronized on a regular basis.

5.4 Test impact of remediation techniques

The relative effectiveness of each form of remediation depends on how centralized date handling is in the system element, the range of dates to be processed, and the time available to complete the project. The choice of remediation techniques has different testing impacts. If date handling is centralized, a relatively small number of subelements may need to be modified and tested. Drastically decentralized date handling, where data is accessed by separate code segments throughout the application, may require replacement.

5.4.1 Replacement (also called migration)

Replacing a system element is likely to require a larger amount of testing to ensure that all functionality, including Year 2000 specific conditions, operates as expected. Once a baseline for current operation has been established, transition tests and future regression tests should also be completed.

5.4.2 Windowing

Testing windowed remediation involves validating that both the windowing algorithms and all data fall within the window. Testing should assure that all dates that fall within a given date window are retained and dates outside that window are rejected.

The logic that allows the century of a six-digit date to be interpreted may need to be implemented at any place where a date is processed. This means that the probability of remediation errors is very high because a large number of changes need to be made in complex areas of the system. All system elements that process windowed dates directly should be considered at risk.

Date windows restrict dates to a 100-year range. A system may have several different date windows that allow different data sets to have separate ranges. This can increase the complexity of testing by causing multiple boundary conditions to occur within a system or system element. Unit testing would require human interpretation of multiple date representations or the construction of reliable test rigs that could make the interpretations for the test engineer.

System elements that rely on windows should be protected against data corruption from input dates outside the window. If restrictions on entering dates outside the input range are enforced internally, the element may be unit tested. Otherwise each data path that brings dates into the window should be checked separately.

5.4.2.1 Fixed

Fixed windows are established in the design of the system and are not accessible or changeable by the user. The testing impacts for this technique are similar to those discussed above. Assure that the correct window was chosen and that its date domain is adequate.

5.4.2.2 Movable

Movable windows are set by the user and are normally fixed at that point. The testing impact for this technique is similar to those discussed above. The user defined window (pivot year) means that the boundaries of the window are not known until the system is installed. The testing necessary for the data source and the window algorithm is complicated by the need to test multiple boundary conditions that represent the possible range of pivot years. It is possible that incompatibilities with other system elements may be apparent only within a specific set of boundaries.

5.4.2.3 Sliding

The sliding window technique bases its pivot year on the system date. Testing should ensure the system's ability to transition from one pivot year to another. Care should be taken to check that dates at one end of the

window are not wrapped to another century when the pivot year changes. The other testing impacts for these techniques are similar to those for movable windowing techniques.

5.4.3 Expansion

Expansion requires data and program modification. Testing the ability of system elements to interpret the new data type should be performed on every element that accesses the data directly. Changes in the input to an element may expand the range of its output beyond the original specification. It is necessary to insure that any required century information is included in the output, and that the element receiving the data can interpret it.

Expansion of permanent data stores usually requires the use of conversion programs. The conversion programs themselves need to be tested for accuracy in updating the entire range of dates in the current data store.

A larger date format may increase both the permanent storage and memory requirements of a system. Performance testing can be used to gauge what degradation may have occurred.

Expansion may require the creation of bridge programs to access existing data that, because of legal concerns, expense, or for audit purposes, cannot be converted to the new date format.

Expansion normally requires modification of the size of dates in existing record formats. If data is accessed via an offset from a specific point in the record, that offset may need to be changed to compensate for the larger date field. In some cases, system elements that are not date sensitive, but which happen to access the same data source, may be at risk. This leads to greater levels of test effort.

5.4.4 Encoding

The encoding technique usually requires changing both data and logic. Like windowing, program logic may need to be tested and, like expansion, other systems that access modified data should be tested to ensure their continued compatibility. Groups of applications that share date-data may be implemented simultaneously or implemented through temporary bridges.

The process used to encode existing data will normally include automated conversion utilities that should be tested.

Encoded data require decoding for testing. This may add additional risk and resource requirements for the test procedure.

5.4.5 Elimination

Elimination is the retirement of a system or application no longer deemed necessary.

5.4.6 Data encapsulation

It is possible to place bridges around existing data to allow their use within a system remediated to use a new date format. It is important to ensure that all access to this existing data is through the bridges. If other system elements access existing date data, expecting it to be in the remediated format, the remediated element may misinterpret the dates or suffer fatal errors. Data encapsulation can eliminate the need for data conversion since conversion is accomplished on the fly. Instead of testing the conversion utilities or the converted data, the bridge is tested to insure its ability to correctly translate the range of possible data. It may be helpful to isolate the bridge system element(s) in order to effectively test. Since this method avoids the date transition, future date testing may be fully or partially eliminated.

5.4.7 System or system element encapsulation

There are several techniques that rely on bridges to isolate existing system elements from date-data outside the system elements working domain. For example, bridges may be created that shift all dates back a preset amount of time allowing the system element to process some post Year 2000 dates as if they were actually prior to Year 2000 rollover.

Assuming that the system processes a range of dates prior to Year 2000 rollover properly, testing should ensure the bridges ability to shift dates in and out of that range accurately. Existing data may need to be modified since dates that were at the beginning of the system's old date range may be shifted out of range by the bridges. Error handling for out-of-range dates should also be checked.

Shifting dates can affect calculations that compute or are computed from the day of the week, leap years, holidays, lunar phases, tidal cycles, and other factors that occur on different dates from one year to the next. All dates that are shifted are at risk, so tests should encompass the entire range of dates.

5.5 Special considerations involving embedded systems

Embedded systems pose numerous specific Year 2000 problems, many of which fall beyond the scope of this document. For example, the white box testing methods discussed involve analysis of programming logic. Embedded systems may be configurable by the user and may be interfaced with other systems or devices, but they may not be programmable by the user. Obtaining access to and interpreting the internal programming of embedded systems may be a complex and time-consuming process. Evaluation of risk and resource limitations may counsel against such efforts.

The black box testing methods discussed may often be more appropriate for embedded systems. Embedded systems may have specialized and circumscribed user interfaces, making it difficult to prepare and input appropriate test data to the system, and monitor and evaluate the associated system output. Embedded systems may function largely through specialized interfaces with other systems, for example, data acquisition and control and communications interfaces. Identification and correlation of the input and output of an embedded system interfaced within a larger, more complex system may not be an appropriate allocation of testing resources in light of the risks posed by a particular embedded system.

Nevertheless, many of the test procedures in Clause 6 may be adapted for use with embedded systems. Examples include testing special-condition dates, date formats, event-triggers, and date interval and arithmetical calculations.

5.5.1 Supervisory distributed control systems (DCS)

Supervisory distributed control systems contain proportional integral derivative (PID) control centers that often operate asynchronous to the supervisory DCS. PIDs do communicate with the supervisory DCS. Alarms from the PIDs are prioritized as follows:

- a) Interrupt driven, for severe alarms, providing occurrence time-date stamps; and
- b) Polled alarms for DCS recognition of critical and noncritical alarms, with process variable time-date stamps.

Communicated data logging, trending, and predictable functions with integral time-date stamps can also be event-driven and aperiodic. The alarms are commonly provided to an alarm summary field and presented to plant operations as an alarm log summary screen, with a printout. Severe alarms are intended, in many instances, to shut down plant operations due to safety concerns. Should a corrupted data packet, time-date alarm occur, plant safety may be jeopardized.

PID controllers and other asynchronous process control loops may contain independent operating systems, firmware, and ladder logic that runs independent of the supervisory DCS control. Independent control nodes support a given process and are self-regulating, based on the process demands. Such loops contain system clocks that cumulate calendar functions based on arbitrary initial settings, input by field personnel. Seldom is time synchronization accurate or provided.

Event-driven time-date inputs of alarms and plant operation trigger points are provided to the supervisory DCS control. During the Year 2000 time-date transitions, the asynchronous and non-Year 2000 compliant time-date stamped data entering the supervisory control could cause fatal errors in programmed time-date driven functions, due to date misinterpretation.

5.5.2 User versus manufacturer testing

End users of embedded systems may find that their testing efforts are limited by a lack of adequate system configuration information, specifications, access to original code and proprietary testing tools, and availability of qualified testing personnel. For recently designed systems, the manufacturer may already have many of these resources in place. For this reason, it is often more cost effective for the manufacturer to test the system. Users of older systems may find that the manufacturer will no longer support the system, making it necessary for the user to test the system or contract for testing with a qualified service provider. Even if Year 2000 certification is available from the manufacturer, it may be prudent for the user to test the specific system in its current configuration. Reasons why a particular system or system element may fail while other apparently equivalent systems do not, include, but are not limited to

- Different components may be used in the assembly of the system tested by the manufacturer than are used in the system as installed. Although two integrated circuits or other components may be specified as functionally equivalent, differences in their implementation may cause one to have a Year 2000 failure while the other continues to function properly.
- Embedded systems are often elements of larger systems. As with other systems, improper interfacing between system elements can cause a failure. A manufacturer may not provide Year 2000 certification of an interface between two system elements either because there is a known incompatibility or because the interface has not been tested.
- Customization of embedded systems is often necessary to make them suitable to a specific installation. They range from minor modifications in control code to additional hardware requiring new functionality in the controller. A customized embedded system may be ordered from the manufacturer or the system could be modified during installation, maintenance, or system upgrades. Changes in hardware or code may cause the custom system to suffer functionality failures that do not occur on the standard system a manufacturer uses for testing.
- Documentation may specify methods for use, including limitations on date formats and ranges that could affect application of the technology in a broader systems context.
- Interchange of data with other systems or components may require use of vendor-defined “proper” formats.

5.5.3 System isolation to accommodate safety concerns

Embedded controls are often used in situations where there is a high risk to property or personnel. To reduce this risk, control systems can often be isolated from the systems they control. This allows the controller to be tested while virtually eliminating the physical risks associated with the system. Generally, this means disconnecting the controller and plugging its inputs into devices that can generate the signals needed for testing and its outputs into a monitoring circuit. Simulation techniques discussed elsewhere in this clause are a good way to accomplish this and reduce the physical risk associated with testing these systems. In some cases it may also be possible to dry run a system or substitute less hazardous materials to reduce the risk of potential damage during testing.

5.5.4 Date simulators

The system time on some embedded systems may be difficult to change. This is particularly true in situations where critical functions are based on time-date information or where synchronization is critical. In certain cases it may be possible to use a date simulator that either replaces the system's RTC or intercepts requests for the system date/time and returns an advanced system date. In either case, it is necessary to ensure that all system date information is obtained from the simulation. If low-level functions are able to bypass the simulation and access the system RTC directly, any tests run might be invalidated. It is also possible that multiple asynchronous RTCs might exist within the system. If one of these RTCs is not included in the date simulation, the presence of two different system dates would, again invalidate testing.

5.5.5 Environment simulation

Simulating some hardware system elements can allow other elements to be tested in an off-line, controlled environment. In general, a simulation of a system element is designed from the functional specification of that element or by specifying and simulating its internal parts and building the simulation from these parts. A functional specification states how the system element is supposed to function, not how it actually performs. This means testing a simulation tests the specification, not the implementation. For this reason, testing of a simulated system element should not be substituted for testing of the actual component. Simulations can be used to create a test environment for other system elements under test.

5.5.5.1 Logic testing

Simulating all hardware in an embedded system may allow initial Year 2000 testing of the system's code. This may be especially useful in situations where there is a possibility of damage to equipment or the environment, or injury to personnel. By isolating the code from the hardware it may be possible to find errors in the program's logic without physical risk. This type of simulation may also be used to allow acceptance tests of a code remediation to be performed off-line.

5.5.5.2 Bench testing

In certain situations it may be possible to remove a system element and test it as a unit. When testing embedded controllers, simple test scaffolding may not give sufficient information for the test engineer to recognize some types of errors. A simulation can be programmed to emulate the mechanical delays inherent in the system's hardware and provide sophisticated displays that make errors apparent to the tester. For example, a controller attached to a robot arm that welds a part onto an assembly might have a delay in its program to allow the part to be positioned by another robot before the welding operation begins. Simple displays of the controller's output might show only the movements of the robot's joints. It might not be apparent if a failure in the delay caused the two arms to collide or caused the welding operation to complete before the part was in position. A simulation that allows the controller to operate a virtual model of the two arms might make this failure apparent without requiring an online test that might damage the robot hardware.

5.5.6 Asynchronous interfaces

Some embedded systems have multiple real-time clocks that are set manually by operators and service technicians. The system design usually allows for these clocks to be slightly out of synchronization. An example would be, if day-of-the-week information is not important to the computations based on one or all of these RTCs, then a six-digit clock could rollover to 00-01-01 and continue to function properly, even without an indicator for the century. If the clocks are slightly out of synchronization and a comparison or calculation is made between date/times from two different clocks at Year 2000 rollover the difference between the date/times could appear to be nearly 100 years. These problems can be intermittent and difficult to track since the error may depend on which clock reaches rollover first and what process is running during transition.

5.5.7 Expansion statement

Expansion of date fields in embedded code may cause information adjacent to storage locations used for dates to be overwritten or corrupted. It is recommended that after expanding date fields, the embedded code be functionally tested to ensure it still performs its intended function.

5.5.8 Test duration statement

As part of test planning, it should be determined how long the subject device, component, or system should be running prior to establishing steady state conditions, how long it should run following a date change, and how long it should run to reestablish steady state conditions. Examples include:

5.5.8.1 Pretest

Steady-state conditions should be established by running the device, component, or system for at least five minutes, or, if the device, component, or system performs a calculation function, then it should run for at least ten calculation cycles.

5.5.8.2 Test condition

Following a date change, the device, component, or system should be run long enough to determine the success, failure, or acceptability of the test. If the device, component, or system performs calculation functions, it should then be run for at least ten calculation cycles. Complex devices or components (those having multiple processors or sophisticated control software that perform error checking and corrections) should run for several hours as experience has shown that it may take an extended time for small errors to build to a critical enough fault for the device or component to fail. If no calculations are performed, the run time following the date change should be based on the complexity of the device, component, or system. Simple devices and components should run for at least five minutes. Systems should also be run for several hours to allow for small errors to build to a critical enough fault for the system to fail. Each test plan should indicate the duration of the run time following a date change and provide some evaluation of its acceptability.

5.5.8.3 Power on/power off test conditions

For power on/power off tests a sufficient time should be allowed to pass to ensure that any capacitors have time to discharge. Physical inspection or vendor information may be used to determine if there are capacitors that need to be discharged.

Another consideration is devices or systems that have periodic update or data clean-out functions. Tests for devices with these functions should be of sufficient duration for the device or system to go through the functions at least once. As an example, if a device does a data update every twelve hours, the device needs to run for at least twelve hours after the test data is input to ensure that the test data is read by the device and the appropriate pass/failure/acceptable deviation determination made.

5.5.9 Sampling statement

The purpose of sampling is to ensure that all devices or systems with the same model number have the same Year 2000 functional performance. Test plans should evaluate the need for a sampling plan when the test subject contains multiple instances of the same model devices or systems. One method for achieving this is to use knowledge or logic-based sampling using knowledge of chip sets and operating systems to group the devices or systems for testing. Alternatively, random sampling methods may also be applied when knowledge of chip sets and operating systems cannot be obtained.

A useful example of knowledge or logic-based sampling is an organization that chooses to test a set of same model numbered devices with nearly consecutive serial numbers by selecting the earliest and latest serial

numbered devices for testing identical behavior. Given these devices exhibit identical behavior, a further decision might be made to sample randomly and test one or more of the devices within the serial number sequence. Given that all tested devices exhibit identical behavior, the organization could then make the decision that all the devices in the original set exhibit the same behavior and results of the performed testing represents results for all the devices in the set.

5.6 Special-condition dates

The following is a list of dates that are recommended for consideration in the validation of Year 2000 remediation. This list is intended to be illustrative, but is not exhaustive. It may be appended or modified for a specific environment.

5.6.1 1900-01-01 (Monday)

The number of days in a century is not evenly divisible by seven, so no two consecutive centuries start on the same day of the week. If an algorithm disregards century information when making day-of-the-week conversions, incorrect results may occur (see also 5.6.8).

5.6.2 1999-01-01 (Friday)

In many systems, the date is parsed into individual year, month, and day variables, and the validity of the date is checked. Often an indicator is needed to trigger logic that reacts to a special situation. If the system insures that the variable is a valid date, a specific year value or date may be used to indicate the special situation. The year “99” and dates within 1999 have been used for this purpose. When the reserved date occurs in normal data, the system may trigger special-condition logic that does not apply to the situation.

5.6.3 1999-09-01 (Wednesday)

The four-digit date format (YY-MM) is sometimes used with the three-digit input 99-9 as a representation of an unknown or “out-of-range” date. The input is interpreted to the full date 99-9-1 and stored. As long as the date is outside the range of normal data it is recognizable as “place-holder” data. When the date 99-9 becomes a plausible entry for the field, it becomes difficult to tell which 99-9 is a real date and which is a placeholder that needs to be replaced with real data.

5.6.4 1999-09-09 (Thursday)

This date is commonly used to indicate an unknown date in six-character (i.e., 99-9-9) data entry fields that don’t require a leading zero. It was chosen because it was easy to type and yet far enough in the future to be easily differentiated from “real” dates. As 99-9-9 nears it will become impossible for the computer or the user to know if the entry is valid or not.

5.6.5 1999-09-10 (Friday)

In systems that have used 9-9-99 as a never-expire date, logic allowing deletion of data after a specified date may fail to protect data that should be maintained forever.

5.6.6 1999-10-01 (Friday)

This is the first day of the U.S. Government Fiscal Year, FY2000.

5.6.7 1999-12-31 (Friday)

The last day that can be represented in standard six-digit date format without Year 2000 rollover risk. This date is sometimes used to trigger special logic. It must be established that the system is able to distinguish between a regular end-of-year 1999 date and a special meaning date. For example, a license key intended to expire on 99-12-31 should not be confused with one that has no expiration date. This is also the start date for most Year 2000 rollover testing.

5.6.8 2000-01-01 (Saturday)

The first day of the Year 2000. There are many issues related to this date.

- A system with a day-of-week function based on six-digit dates may change from Friday, 1999-12-31 to Monday, 2000-01-01 at Year 2000 rollover. 1900-01-01 was a Monday. A day-of-the-week error could occur on any date after Year 2000 rollover if the calculation to derive the day of the week assumes that all dates have years between 1900 and 1999.
- There is a possibility that the date will be misinterpreted as 1900-01-01.
- Systems date counters may increment to erroneous dates like 19100-01-01.
- Parsing functions may misinterpret dates entered with one or both leading zeroes omitted (see 5.6.12).

5.6.9 2000-01-03 (Monday)

This may be the first business day of the Year 2000. Certain business software calculates using proper business dates and days. Consequently, failures associated with this software would occur at this point. This day may be a holiday, not a business day. It is special from that perspective in that it would normally be a business day, but may not be in this case. It would be the first holiday of the Year 2000.

5.6.10 2000-01-04 (Tuesday)

This may be the first business day and first banking day in the Year 2000. Business applications may be sensitive to calculations using proper business dates and days. Consequently, failures associated with this software would occur at this point.

5.6.11 2000-01-07 (Friday)

This is the first Friday in the Year 2000.

5.6.12 2000-01-10 (Monday)

This is the first seven-digit date after Year 2000 rollover if leading zeros are not used for day and month representations. New or modified parsing functions required by changes in date-input formats or for interpreting an extended input range may fail when the number of digits representing the day changes. Parsing functions may need to be tested with all possible combinations of dates with one- and two-digit month and year values and combinations including and omitting leading zeros. (Examples: 2000-1-1, 2000-01, 2000-01-1, 2000-01-01, 2000-1-10, 2000-01-10,...) Each acceptable date representation should be checked to insure that it is correctly translated into the system's internal representation.

5.6.13 2000-01-17 (Monday, Martin Luther King, Jr. day, U.S. holiday)

This may be the first Monday holiday in the Year 2000. This is a holiday that is always celebrated on a Monday rather than on a specific date in the U.S. A day-of-the-week calculation may be required to identify this

date as a holiday. In other countries a similar situation may exist for locally celebrated holidays on other dates. These dates may need to be tested instead.

5.6.14 2000-01-31 (Monday)

This is the first month-end day in the Year 2000. Many programs key on month end as a trigger for a periodic function and may fail here.

5.6.15 2000-02-28 (Monday)

This date is not expected to cause any specific Year 2000 errors. Its relevance to testing is that it should be used as a start date in testing the system's ability to increment to 2000-02-29. This is necessary because it is possible for a system to recognize 2000-02-29 as a valid date when the date is manually set, while a separate system element increments the date directly from 2000-02-28 to 2000-03-01.

5.6.16 2000-02-29 (Tuesday)

The Year 2000 is a leap year. Program logic used to identify leap years may be incomplete. This could cause date processing errors for the remainder of the year.

5.6.17 2000-02-30 (Non-existent)

This day does not exist. Date functions should continue to recognize this as an invalid date.

5.6.18 2000-03-01 (Wednesday)

This is the first day after leap year day. The date calculations that transition from the last day of leap year February to the first day of March could fail. The possibility exists that some part of a system may fail to recognize Year 2000 as a leap year, which may lead to a condition where dates are no longer synchronized well, as day-of-the-week offsets can occur. This is also the first day of the last month in the first quarter of the Year 2000.

5.6.19 2000-03-31 (Friday)

This is the last day of the last month in the first quarter of the Year 2000. Quarter-end dates are significant in business and financial applications.

5.6.20 2000-04-03 (Monday)

This is the first business day of the second quarter of the Year 2000.

5.6.21 2000-04-17 (Monday)

This is the Primary U.S. Income Tax due date in the Year 2000.

5.6.22 2000-04-28 (Friday)

April is the first business month whose last day coincides with a weekend in the Year 2000. This is the last business day of April.

5.6.23 2000-04-30 (Sunday)

This is the first month-end that coincides with a weekend in the Year 2000.

5.6.24 2000-06-30 (Friday)

This is the last day of the last month of the second quarter. This is the half-year point as well as the end of the fiscal year, or business cycle, for many businesses. This is the last business day of the first quarter end in the Year 2000, which coincides with a weekend.

5.6.25 2000-09-29 (Friday)

Last business day of the third quarter in the Year 2000.

5.6.26 2000-09-30 (Saturday)

This is the last day of the government fiscal year and last day of the third quarter of the Year 2000.

5.6.27 2000-10-01 (Sunday)

This is the first seven-digit date with a two-digit month value. Parsing functions may need to be modified to allow for new date formats and a wider range of date-data during the remediation process. If leading zeros are not required for date-input, the date value entered might change the placement of numeric values for month, day, and year within the input string. A parsing function that doesn't allow for this input variation might interpret 2000-10-1 as 0200-01-01 or might interpret the punctuation as part of the day or month value.

5.6.28 2000-10-10 (Tuesday)

This is the first date after rollover that must be represented as an eight-digit date. Parsing functions may fail when the number of digits changes. This is similar to failures discussed in 5.6.27, but could result from mis-interpretation of either the month or year value.

5.6.29 2000-12-31 (Sunday)

The last day of the second millennium of the Gregorian calendar. The ordinal date 1900-365 was the last day of 1900. Since 2000 is a leap year, its last day is 2000-366. An incomplete algorithm for determining the length of the year might cause an ordinal-based system to transition into the new millennium a day too early.

5.6.30 2001-01-01 (Monday)

This is the first day of the third millennium on the Gregorian Calendar. There is a possibility of errors in computing the day of the week. This is also a holiday and not a business day.

5.6.31 2001-01-05 (Friday)

This is the first Friday in the year 2001.

5.6.32 2004-02-29 (Sunday)

First leap day after Year 2000 rollover not affected by a century or millennium transition.

5.6.33 2004-12-31 (Friday)

This date can be used to determine if normal leap years are recognized by an ordinal-date system.

5.7 Special date conditions

5.7.1 Out-of-range

Any real date that cannot be represented in the system’s or system element’s internal date format is considered an out-of-range date. Out-of-range dates might include dates outside a system element’s date window, or before the start date of an integer offset date representation. The range of dates used will depend on the system implementation. They may be used in testing to determine the system’s ability to protect data from corruption. If not recognized and rejected these dates may be wrapped into the acceptable range or misinterpreted in some other way, and introduced into the system. This could lead to data corruption.

5.7.2 Nonexistent dates

Dates with field errors, such as a month value of 13 or the 31st day of a 30-day month should be included in test data. This should be done to ensure that the system retains its ability to recognize these dates, reject them, and recover gracefully.

Common field error dates	
0000-xx-xx	2000-04-31
1999-02-29	2000-06-31
2000-xx-00	2000-09-31
2000-00-xx	2000-11-31
2000-02-30	2000-13-xx
xx – Indicates a “don’t care”; any value in these fields indicates a nonexistent date	

In some systems, nonexistent dates may be used to indicate a special condition such as an unknown or out-of-range date. If allowances are not made for the continued use of these dates the remediated system may reject or misinterpret them. This could result in a loss of system functionality. Tests for indicator dates should return a predictable result in accordance with the systems specifications.

Common indicator dates	
0000-00-00	2000-00-00
1999-99-99	

5.7.3 Flag codes

Dates that have memorable patterns such as 99-9-9 or 99-1-1 may be used by some systems to override normal date processing and start a routine to handle specific condition flags.

5.7.4 Year 2000 rollover

The transition between 1999 and the Year 2000 may affect a date counter’s ability to increment properly, causing an error at the moment of transition. Comparisons and computations based on dates may give erroneous results if operations span rollover.

5.7.5 Leap year conditions

In the Gregorian calendar, 1900 was not a leap year, but the Year 2000 is. A system or system element that represents both years as 00 may process both years as a leap year or as a non-leap year. The ability of a system to recognize 2000-02-29 and not 1900-02-29 affects calendars and date displays as well as computations using a span of time including either of these dates.

5.7.6 Date comparisons using inequalities

It is often necessary to determine which of a pair of dates is prior to the other. If the date values being compared only include the two least significant digits of the year and they fall on opposite sides of the Year 2000 rollover, the operations used to compare them may return inaccurate results.

For example:

The four-digit comparison gives the correct result

$$2000-02-15 > 1999-12-15 = \text{true}$$

while its two-digit equivalent gives an incorrect result

$$00-02-15 > 99-12-15 = \text{false}$$

5.7.7 28-year repetition

Any two dates between 1900-03-01 and 2100-02-28 that are 28 years apart will have the same day of the week associated with them. In cases where the results of tests are related to the day of the week, running the test cases twice with all dates 28 years apart allows the results to be compared for equivalence. This can help to eliminate the need to calculate test results by hand for these tests.

5.7.8 Computed day intervals

Time intervals may be computed by subtracting one date from another. If the operands for the subtractions are dates with only a two-digit year, the computation may give incorrect results when the date range includes the Year 2000 rollover.

For example:

The four-digit comparison gives the correct result

$$2000-03-01 - 1999-12-31 = 61 \text{ days}$$

while its two-digit equivalent gives an incorrect result.

$$00-03-01 - 99-12-31 = -36\,464 \text{ days}$$

5.7.9 Increment and decrement date by a count

Computing a target date a specific number of days prior to or after a given date may fail when the target and given dates would be on opposite sides of the Year 2000 rollover. Adding or subtracting enough days to cross the Year 2000 rollover may create a two-digit year value that is either negative or greater than or equal to 100. The system may not be able to convert these dates into standard date formats reliably.

5.7.10 Day-of-the-week numbers

It is generally easier for a computer to manipulate numbers than strings, so day-of-the-week strings are often enumerated and referenced by the associated number. The day-of-the-week number can be calculated by computing the interval between the date in question and a reference date and taking the result modulo seven. As noted above, the ability to compute date intervals is at risk, so the results of day-of-the-week calculations based on them is also at risk.

5.7.11 Ordinal dates

When an ordinal date format is used, a leap year calculation based on the year value determines if the date XXXX-366 (where XXXX represents the year) exists in a particular year. Errors in identifying a year as leap year affect calculations of time intervals and conversions to other date formats.

5.7.12 Windowing dates

System elements that use date windows have defined boundary conditions that should be tested. Boundaries of windows may be different from the valid date interval of the system and should be specifically tested.

5.7.13 Day of the week

Calculation of the correct day of the week may involve logic to identify leap years, convert one or more date formats into day-of-the-week numbers, interpret date windows or any number of other errors mentioned elsewhere in this list.

5.7.14 Calendar date to ordinal (Julian) conversion

Failure in these types of procedures is usually related to misinterpretation of leap year causing the resulting date to be off by one.

5.7.15 Local date formats

Systems that have multiple date display formats usually have an internal date representation that is converted to a formatted string for output. A separate function may be used to convert from the internal representation to each individual local date format. Errors can occur because the internal representation doesn't contain enough information to distinguish dates in Year 2000 from earlier dates. It is also possible that individual conversion functions may not be able to interpret the larger input values that represent Year 2000 dates.

The internal date representation of a system is usually set from an RTC using BIOS calls to transfer date-data from the RTC to the system clock. For globalization purposes, a system may represent an external date in a locally used calendar while one or all of these internal system elements may represent dates in the Gregorian system. Any one of the internal elements or the system element that converts the system date to a local date format may fail when the RTC encounters Year 2000 rollover. This can happen whether or not the local date format has any corresponding date transition.

5.7.16 Daylight-saving time ("summer time")

Daylight-saving time changes occur on the first Sunday in April and the last Sunday in October in the U.S. Other countries may have different dates for initiating and terminating "summer time." This means that systems using time standards that include daylight savings must use day-of-the-week information to define if a date falls within the affected period. As described elsewhere in this list, day-of-the-week calculations may fail during Year 2000.

5.7.17 Year and month extraction values

The methods for extracting the day, month, or year value of a date is dependent on the storage format of the date. Each format has its own potential problems.

5.7.18 Time zone offsets

Information passed between system and system elements is often associated with a time stamp. In Wide Area Network environments the receiving system may be in a different time zone than the originating sys-

tem. This creates a situation where one system may rollover into Year 2000 while the other continues to operate in 1999 for several hours. If the time stamps include only two-digit year values the receiving system may assume the information is invalid since the time stamp associated with it is either 100 years old or 100 years in the future.

In some cases, a system may use a single time reference, such as Coordinated Universal Time (UTC), ZULU time, or Greenwich Mean Time, for all system elements regardless of their local time zone. This may be hidden from the user by converting date/time groups to local time for I/O purposes. A failure can occur when the offset between system time and local time crosses Year 2000 rollover.

It should also be recognized that not all time zones are an integral number of hours, and conversion algorithms that use data that spans different time zones must be prepared to deal with these nonintegral time zones. Examples of countries that contain such zones are parts of Australia, India, parts of Canada, Iran, Nepal, Sri Lanka, and some Pacific Islands. Further, some Pacific Islands have a shift to daylight-saving time that is not a full hour and that also must be taken into consideration if the data is from or is intended to be transmitted to such places.

5.8 System components to be tested

Effective system testing may involve the creation of test cases that act on or require various combinations of the following system components. The test cases presented in the following subclauses should be considered in the context of each component.

5.8.1 System element

System elements may be tested before and after remediation. This can include, but is not limited to, software, hardware, and firmware. Test cases should validate that no desired preexisting functionality has been lost (regression testing) and that new “Year 2000” functionality works as expected (compliance testing).

5.8.2 System data

Systems should be tested using data having current dates and dates that have been advanced beyond the Year 2000. Date-data that have been advanced beyond the Year 2000 may be referred to as aged data. Existing data can be aged or Year 2000 data set can be created to meet testing requirements. Systems should also be tested with the system date in or post-Year 2000 with data from before Year 2000.

5.8.3 System time

Systems may be tested using current system time and with the time set beyond the Year 2000. Setting a system time to beyond Year 2000 may allow the testing of many system factors not normally observable using current time. These factors include the operating system utilities that use internal time stamps, system archiving or backup utilities, and other aspects of the internal operation of a system.

System time is usually maintained in a hardware system element that is protected against unintentional changes to its value that might be caused by a system reset or a local power outage.

This standard refers to the time source as an RTC to distinguish it from the system clock that synchronizes the internal processor functions. The RTC may be read directly, but more often its contents are stored in another clock register that is incremented separately from the RTC. The two times may be synchronized at specified intervals or only when the system is started. An example is the X86-based PC that has a battery-backed RTC. At start up the RTC is used to determine the date and time. These values are reformatted and placed in memory. From then on, the time and date are incremented by the System BIOS. Some operating

systems may maintain another clock that is updated by the operating system itself. Changing the system time requires re synchronization of the clocks.

Distributed systems, including networks and plant automation systems, may have multiple RTCs residing in many individual components. The concept of system time assumes that these clocks are synchronous or nearly synchronous. This may be accomplished by automatically setting each individual RTC to the value of a designated master RTC. In asynchronous systems, individual RTCs may be set manually, introducing an error to the system time that might be several minutes.

Some systems do not have or need a system time. This may be true of countdown timers, calculators, simple logical controllers, and other systems or system elements that perform in exactly the same manner, regardless of the date or time. **Caution should be used in identifying these systems since there is a possibility that a timer based on Gregorian dates may be used to implement timing functions even though there is no external reference to the date.**

5.8.4 Data and time combinations

Test cases should represent all combinations of system time and date-data that are possible during the system's transition into the Year 2000.

- System time prior to Year 2000 with data before Year 2000
- System time prior to Year 2000 with data after Year 2000
- System time post-Year 2000 with data after Year 2000
- System time post-Year 2000 with data before Year 2000

6. System elements at risk

This clause is an attempt to outline example system elements or features that have a potential of being affected by date-based algorithms and, therefore, could fail if not designed to cross the Year 2000 rollover. Special condition dates and example test assertions are included that are generic in nature and correlate with certain system elements and that can be customized for specific needs.

6.1 Interfaces: Shared control blocks/API/DDE/OLE

6.1.1 Definition

Application program interface (API): The interface by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the implementation.

Dynamic data exchange (DDE): A protocol that allows application programs to communicate using a client-server model. Whenever the server modifies part of a document that is being shared via DDE, one or more clients are informed and the modification is included in the copy of the data on which they are working.

Object linking and embedding (OLE): Allows an editor to place part of a document into another editor and then re-import it.

6.1.2 Rationale

In transferring date information, these functions may use an intermediate date-data form, which might not support proper handling of century date information. It is necessary to compare source information to destination information to ensure accurate date-data transfer.

6.1.3 Related elements

— None

6.1.4 Test cases

6.1.4.1 OLE

Change of a date field in an application results in a corresponding change in a linked application.

I Test objective

Verify that changing a linked high-risk Year 2000 date that is in a master application (e.g., spreadsheet, database, and presentation) will change the corresponding date in a linked slave application.

II Test conditions

Two applications with linked date-data. Have a master application with a date field that is linked to a corresponding field in a slave application.

III Procedure/results

#	Test procedure	Expected test results
1	Master—Change the date field in application to one of the high-risk Year 2000 dates.	
2	Slave—View the date field in second application that has an OLE link.	The date-data field will be updated to high-risk Year 2000 date.

6.2 Archiving/restoring

6.2.1 Definition

Archiving/restoring: The transfer of data files to and from a long-term storage device, for example, magnetic tape.

6.2.2 Rationale

Most archives specify a retention date or interval. Archiving systems must be able to accurately compute and compare these dates through Year 2000 rollover and beyond 2001. If the dates used don't include century information, an archive recorded in 1997 with a retention interval of five years would be dated for deletion in 2002 and might be considered 95 years old at its creation.

The problem is compounded by the use of certain dates as special logic indicators. In some applications, year fields with the values 00 or 99 are used as indicators for records that have no expiration. Without modification, these applications might save all archives dated for deletion in 1999, 2000, or on specific days in 1999 and 2000 indefinitely. Depending on the retention duration, some systems may already be storing archives that should be deleted when their retention date has passed along with archives that use the same date value to indicate that they should never be deleted, leaving no easy way for the system to distinguish between them.

It is also necessary to insure that the remediated system can restore archives that existed before remediation. This may be accomplished by converting legacy archives to the remediated date format as part of the reme-