

Evaluating Systems for Year 2000 Compliance

Thomas C. Royer
September 30, 1997

ABSTRACT

The evaluation of software products for Year 2000 (Y2K) compliance requires careful planning and a thorough investigation of the capabilities or functions the product provides its users.

The evaluation begins with a bounding process that establishes the product's limits (single application, system, system-of-systems, etc.) and its interfaces to the outside world. Once these limits are established, compliance goals can be determined, critical dates can be identified, and the actual compliance checking can start.

Compliance checking takes place in a number of phases as more information about the product is gathered. System monitoring (actually observing the product in operation) can be used when the evaluators are not sure of the product's date dependencies. Checklists are particularly useful during evaluation to ensure completeness and to facilitate the sharing of Y2K information among several related evaluation efforts in a consistent manner.

In the case of commercial-off-the-shelf (COTS) and Government-off-the-shelf (GOTS) products, special evaluation procedures are required because of the lack of internal design and implementation information. If many products are to be evaluated in a short time, a prioritization scheme can be used to ensure that critical components are checked first.

Automating the evaluation process can produce savings in resource utilization, but will require the use of an *oracle* to predict the product's response to test cases. In some cases, the unmodified (questionably compliant) product can be used as the oracle for the renovated version.

Additional Year 2000 information can be found at:

<http://www.mitre.org/research/y2k>

Table of Contents

Test and Evaluation3

What are We Evaluating?.....3

Setting Compliance Goals4

Proper Processing of Dates Before and After 01/01/20004

Recognize Y2K as a Leap Year.....5

Display, Print, Input, and Storage Dates are Unambiguous6

Dates for “Non-Date Functions”7

Checking for Compliance8

Evaluating for Compliance.....9

Inspecting for Compliance.....9

System Monitoring.....9

Using a Compliance Checklist.....10

Evaluating COTS/GOTS..... 10

Evaluating COTS/GOTS Without a Specification.....11

Evaluating Many Products Quickly.....11

Basic Test Cases..... 12

Automating Evaluation..... 13

Layered Evaluation 14

Isolating Systems for Evaluation..... 15

A Caution Before Testing..... 16

External Testing Facilities..... 16

Summary 16

APPENDIX – Common Date-Dependent Non-Date Functions of COTS/GOTS..... 18

APPENDIX – The Department of Defense Year 2000 Compliance Checklist..... 19

Test and Evaluation

Before we can test or evaluate anything, we must clearly understand what the words “testing,” and “evaluation” mean and how they can best be used in the context of Year 2000 (Y2K) compliance.

The conventional definition for software testing is that it “is the execution of a software product for the purpose of detecting errors.”

When a product is being developed, that is a valid, if limited, definition. But within the Y2K context, we will change it somewhat, while, at the same time, introducing a definition for “evaluation.” These definitions are:

Evaluation is (1) the determination of the risks that the product under consideration will not adequately support the desired user capability or mission; (2) the review of these risks for acceptability; and (3) if the risks are not acceptable, the determination of a mitigation strategy.

Testing provides information that permits the risk evaluation to be done. Testing to support Y2K evaluation may identify errors in the product and, in that sense, it is consistent with the conventional software testing definition.

The Y2K evaluator for a system should keep in mind the definition of evaluation. If it is determined that a product will adequately support the desired user capability (with an acceptable degree of risk), the the product can be considered Y2K “acceptable” even if it does not meet the strict definition of Y2K compliance (see page 4, below).

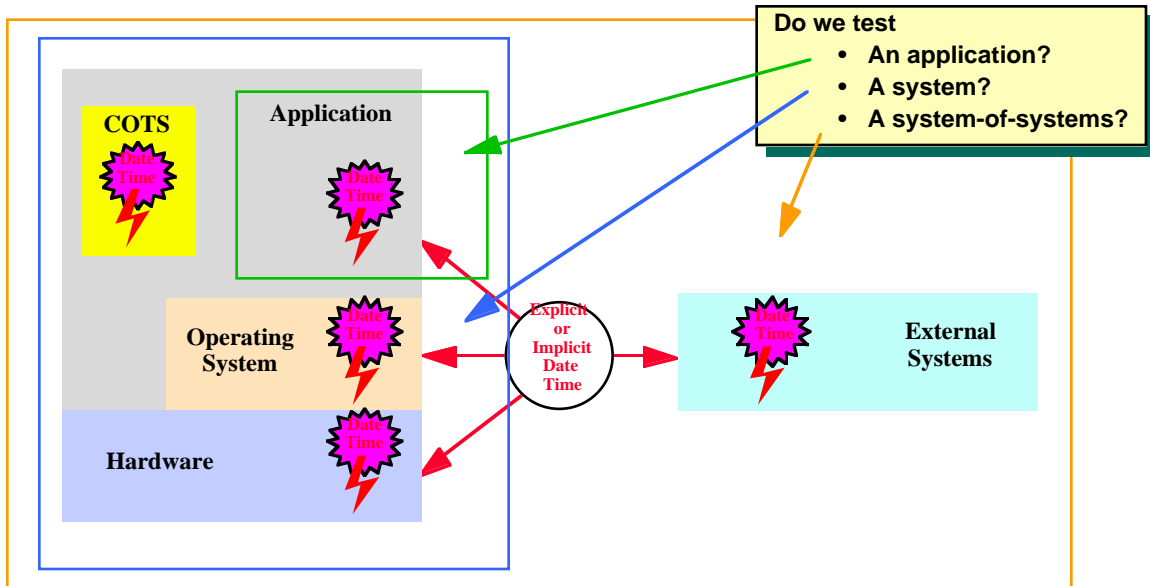


Figure 1—Establishing What We’re Testing

What are We Evaluating?

The first step in doing a product evaluation is the establishment of the limits of that product: exactly what are we evaluating? The answer could be:

- A single application;
- A COTS or GOTS product;
- A “system” composed of several applications and COTS/GOTS products;
- A system-of-systems.

As illustrated in Figure 1, the above list is, more or less, in ascending order of complexity, and the more complex the product being evaluated, the more difficult the evaluation itself becomes. For planning purposes, it is important to identify *all* aspects of the product to be evaluated while ensuring that nothing additional is included in the effort. It is also necessary to identify relevant components that are *not* being tested so that the organization or agency that is responsible for them can be determined.

Setting Compliance Goals

Once we have defined the bounds of the product, we need to know what it is supposed to do. In the general case, the product's behavior is documented in a *specification*. In the case of Y2K evaluation, the specification is the definition of Y2K compliance.

For a product to be Year 2000 compliant,¹ it must:

- Correctly process dates before and after the year 2000;
- Recognize the year 2000 as a leap year;
- Accept and display dates unambiguously; and
- Correctly process logic dates that are used for “non-date functions.”

These are very high level functional requirements, and their meaning must be examined in detail.

In addition, the product must operate properly and provide correct and unambiguous date information as it encounters other “critical date transitions.” Critical date transitions are those for which evidence, either hard or circumstantial, exists that date-related process might encounter error. Some examples of critical date transitions are

- 31 December 1999 to 1 January 2000—the Y2K transition date;
- 28 February 2000 to 29 February 2000—the Y2K leap-year transition;
- 29 February 2000 to 1 March 2000; and
- 31 December 2000 to 1 January 2001.

The above are the minimum set of critical date transitions for which tests will be required for all products being evaluated. Other critical date transitions may exist for specific products such as some financial systems, or others that work with fiscal years. These may be date sensitive to the fiscal year rollover. For systems that follow the Government fiscal year, the following additional date transitions should be considered

- 30 September 1999 to 1 October 1999—the fiscal Y2K transition date; and
- 30 September 2000 to 1 October 2000.

Systems that follow different fiscal years should adjust the above dates accordingly.

Proper Processing of Dates Before and After 01/01/2000

For a product to be Y2K compliant, it must correctly process dates before, during, and after 1 January 2000. This means that:

- From the time the product is declared Y2K compliant until 1 January 2000, it must correctly process dates that are on either side of the millennium point;
- After the century transition to 1 January 2000, the product must continue to correctly process dates on both sides of the millennium point; and
- The product must correctly process the transition itself (Midnight, 31 December 1999).

This span of date cognizance for a product is illustrated in Figure 2.

¹ As stated in the Federal Acquisition Regulation, released August 22, 1997.

We differentiate execution before the Y2K transition (i.e., the passing of Midnight, New Year's Eve, 1999) and after because proper date processing is often dependent upon the current date as maintained by the operating system or some other "infrastructure" component. If the date is correct prior to the transition but incorrect afterward because of a flaw in the infrastructure, correct processing before 2000-01-01 will be followed by erroneous processing afterward.

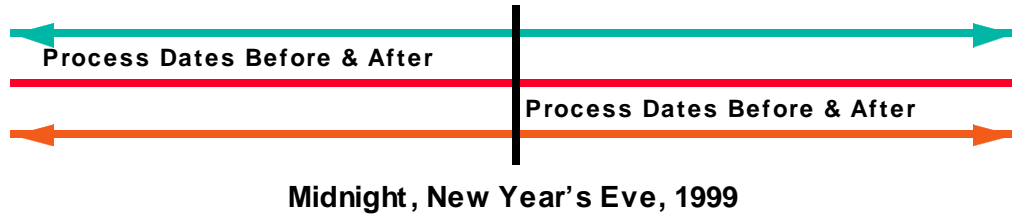


Figure 2—Processing Before and After 01/01/2000

When defining proper date processing, the evaluator must determine the subject system's *date horizons*. The *leading date horizon* determines how far into the future a product may be looking at any given time. Likewise, the *trailing date horizon* defines its interest in the past. For example, a manpower planning program that estimates staff requirements for the next six months has a leading date horizon of six months; a productivity analysis system that calculates a corporation's demonstrated productivity for the past month has a trailing date horizon of one month. When planning test cases, the tester will want to consider the situation where the leading date horizon crosses each of the identified critical dates (e.g., 2000-01-01), the situation when current time crosses the critical date, and the situation when the trailing date horizon crosses the critical date. Figure 3 shows each of these cases when the critical date is the Y2K boundary.

To fully evaluate Y2K compliance, the evaluator will also need a precise definition of "correctly process." Here, the requirement is that the product:

- Correctly convert YYYY-MM-DD format from and to the required display formats and provide correct day-of-the-week and day-of-the-month to applications
- Provide correct date arithmetic services (e.g., days between, etc.)
- Correctly access and store data elements that are keyed by date fields.

Recognize Y2K as a Leap Year

To be Y2K compliant, the product being evaluated must recognize the year 2000 as a leap year (and recognize 1900 as a non-leap year, if 1900 is relevant to the product's operation).

The rule for determining whether or not a year is a leap year is composed of three parts:

1. If the year can be divided evenly by 4 it is a leap year, unless
2. It can also be divided evenly by 100, in which case it is not, unless
3. It can also be divided evenly by 400, in which case it is.

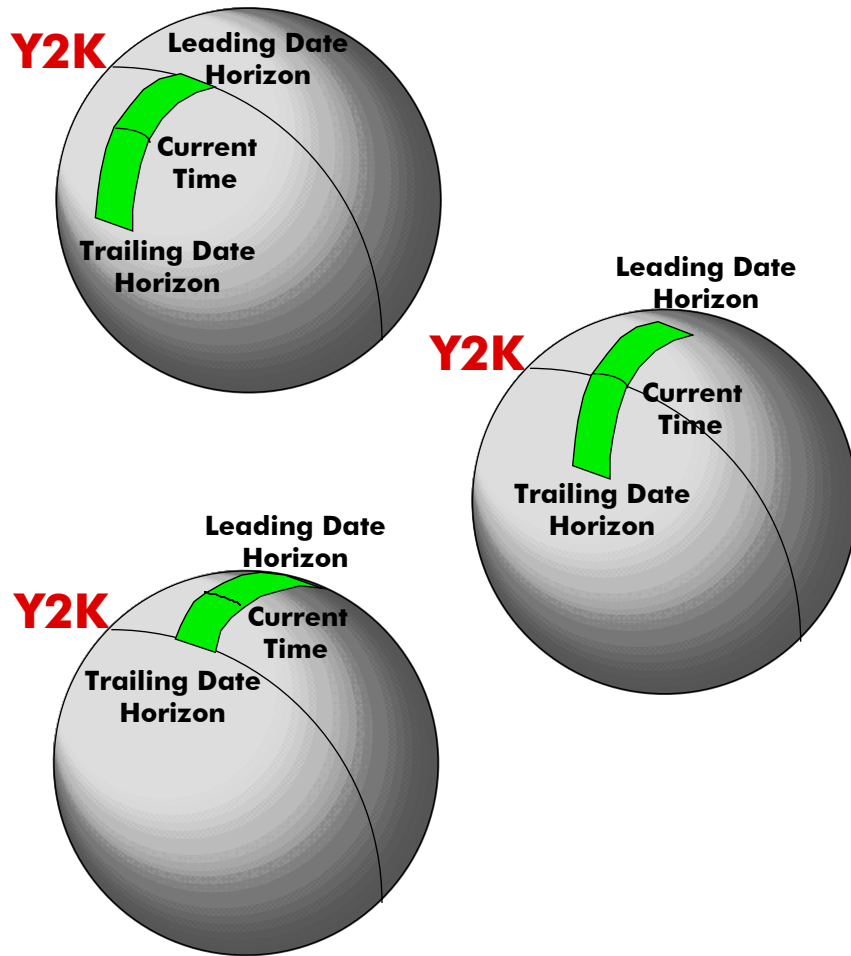


Figure 3—Testing Date Horizons

This is a concern because many older software products have implemented only the first two of the three parts.

Recognizing the year 2000 as leap year requires:

- Recognizing the date 2000-02-29 as valid;
- Recognizing that Julian date 00060 (sixty days from the start of the year) as 2000-02-29;
- Recognizing that Julian date 00366 as 2000-12-31;
- Calculating the number of days between 2000-03-01 and 2000-02-28 as 2;
- Etc.

Display, Print, Input, and Storage Dates are Unambiguous

To be Y2K compliant, the product being evaluated must receive, display, print, and store dates unambiguously.

This means that the product must accept date-oriented input and interpret it correctly. It must provide date-related data to the outside world in a format that is at once acceptable, understandable, and correct. And it must display (including print) date information in a fashion that is understandable and unambiguous to its users.

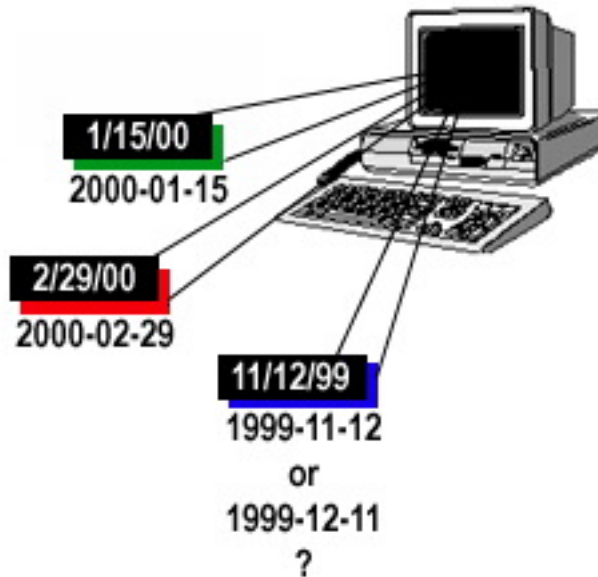


Figure 4—Dates Must Be Unambiguous

In Figure 4,

- The date 1/15/00 may or may not be ambiguous depending upon the date span that the product and the users understand. For most military systems, the year 1900 is irrelevant, and so the date 1/15/00 can be unambiguously interpreted as 2000-01-15.
- Assuming correct internal processing, the date 2/29/00 is unambiguous because 1900 was not a leap year while 2000 is. This date is 2000-02-29.
- The date 11/12/00 may be ambiguous, but not necessarily because of the year designation. Rather, we cannot be sure whether the indicated date is 2000-11-12 or 2000-12-11. Of course, if there is a rigidly defined display order (either month first or day-of-month first), then the display may be unambiguously interpreted (2000-11-12 in the first case, 2000-12-11 in the second).

Dates for “Non-Date Functions”

To be Y2K compliant, the product must properly use dates for “non-date functions.”

A “non-date function” is defined as a function performed by a system or component that, by its nature, does not require date information to satisfy its requirements. Many products, however, utilize dates and times to assist or control the execution of non-date functions.

Some examples of non-date functions that may use date information are:

- Information archiving functions. The product may use the date and time to control the periodicity of data archiving.
- Naming conventions. To insure uniqueness, the product may use the date and time when generating the names of internal or temporary files.
- Passwords. The product may use the date and time in the generation of passwords for users or when considering the validity of those passwords.
- Hashing. The product use the date and time in the generation of hash codes.
- Random number generators. The product may use the date and time as the seed for random number generation.

Not all products perform all of the above functions, and many products use dates for other non-date functions. Examination of the design documentation and the actual code is the

only sure way to identify all pertinent functions like this. For products for which no design or code is available, the tester must resort to the user requirements and to experience. For COTS/GOTS products, the user manual may be of help. The appendix on page 18 provides a partial list of non-date functions for some categories of COTS/GOTS that are frequently implemented in date-dependent ways.

Checking for Compliance

The steps required to make a system or product Y2K compliant are:

1. Awareness;
2. Assessment;
 - a. Impact Analysis
 - b. Inspection;
 - c. System monitoring;
 - d. Evaluation;
3. Correction;
4. Testing/Validation;
5. (Re-) Fielding.

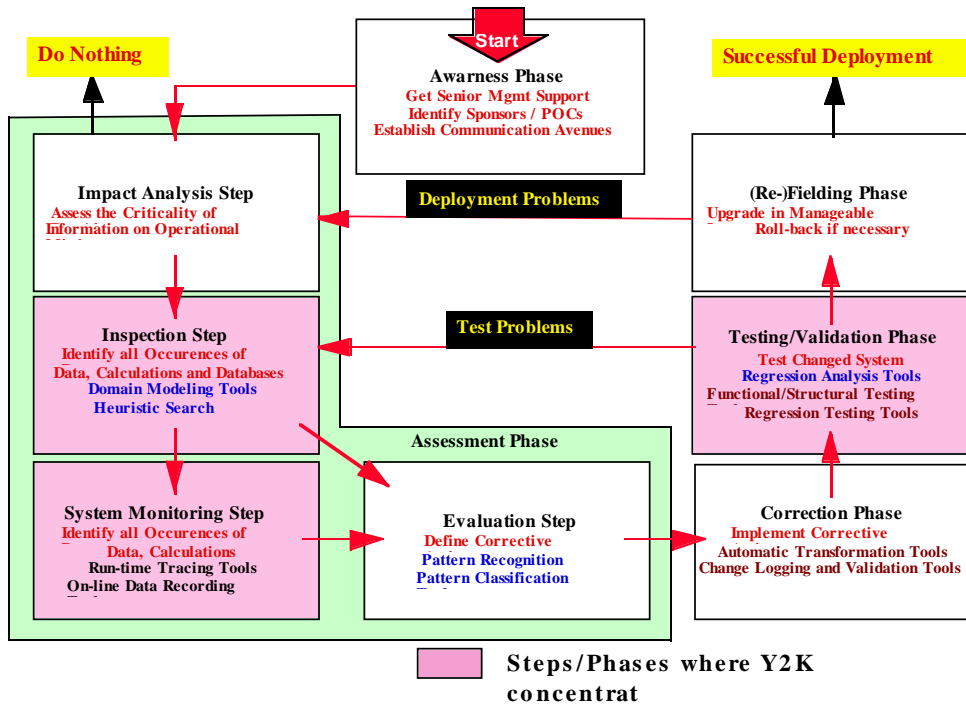


Figure 5—Checking for Compliance

These steps, shown in Figure 5, have been commonly advocated by the software and system community, except for 2c, System monitoring, the function of which will be discussed shortly.

Of these phases, those where the tester/evaluator can play the greatest role are 2b, Inspection, 2c, System monitoring, and 4, Testing/Validation. Each is discussed on the following pages.

Evaluating for Compliance

As shown in Figure 5, the three Y2K compliance steps or phases most conducive to evaluation are Inspection, System Monitoring, and Testing/Evaluation.

During the Inspection step, all available documentation is examined to determine the nature of the product's date and time processing. This documentation includes:

- Specifications which identify date processing requirements;
- Design documents which, if current, identify date-processing algorithms and other algorithms that may use the date;
- Code for software products which will show the specifics of date processing and which, in the case of inadequate design documents, may provide the basis of design recovery;
- User documentation, which may be the most up-to-date documentation available, will identify date processing that directly supports the user or derives from the user.

The System Monitoring step is used to establish date usage and processing when such is not clear from available documentation.

During the Testing/Validation phase, the product is subjected to a series of test cases designed to validate the correct use and processing of date information by the product.

Inspecting for Compliance

When we inspect a system for compliance, we utilize the user requirements, the design specifications, and the actual system code.

The user requirements tell us what the system is supposed to do. If the user requirements involve dates and date processing, changes may be required to attain Y2K compliance, and we have identified an area where we must test explicitly for compliance. However, the user requirements do not specifically identify the component or module that actually does the date processing and often does not discuss date processing in support of "non-date functions." For this information, we must turn to the design specifications.

The design specifications will show how the designer of the product intended it to comply with the user requirements. If these specifications are current, we can use them to identify the specific components or modules that do date processing. From these we can go directly to the code that implements the design. Additionally, the design specifications will be a source for test cases that supplement those from the user requirements.

If the design specifications are inadequate or out of date, it may be possible to recover design information from the code. MITRE has done considerable research into reverse engineering techniques that can provide this kind of information.² Design information is then used to generate test cases for use in the validation phase.

System Monitoring

If date processing is suspected but cannot be verified either because of inadequate design documentation, missing source code, or both, it may be possible to extract date processing information by actually observing the product in operation (either real world, or in a laboratory situation).

² Evaluators should be cautious when using reverse engineering to evaluate COTS products. Many COTS licenses explicitly prohibit reverse engineering to derive design information.

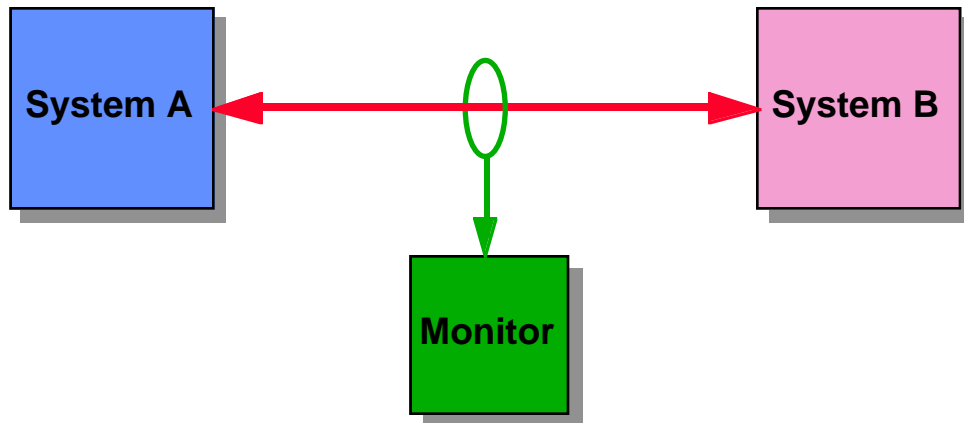


Figure 6—Determining Date Processing by System Monitoring

System monitoring can consist of simple manual observation, or, as shown in Figure 6, it can involve the actual attachment of a monitoring device to the interface or interfaces into and out of the product.

If a monitoring device is used, the natural question is, “Where does it come from?” In many cases, it will have to be specifically developed and the problem of validating the monitor will have to be addressed.

Using a Compliance Checklist

As part of a Y2K validation effort, it is useful to survey the product in an comprehensive way to determine the full scope of the correction and evaluation effort. The best way to do this is with the help of a *compliance checklist* that has been developed independently of the product development activity. The U. S. Air Force uses the use of the DOD checklist (which originated with the Air Force) that is reproduced in the appendix on page 19. It provides a DOD-wide means of documenting and reporting on a product’s Y2K status in the areas of

- Year 2000 processing
- Indirect date usage
- Leap year
- Internal date processing
- External system interfaces
- Date field types
- Year 2000 testing
- COTS/GOTS components

A checklist like this helps ensure a thorough analysis of the product and also makes its Y2K status available to the users of interfacing products in a format that is consistent with the analyses of other products. A checklist is *not*, however, a specification. Evaluators should not take the attitude that they will evaluate a product in accordance with or against the checklist. The checklist is used to ensure thoroughness and consistency in the evaluation; it does not specifically indicate what constitutes compliance.

Evaluating COTS/GOTS

When a product is developed in response to a specific acquisition requirement or as part of an acquisition contract, the evaluator has control over or, at least, knowledge of the design and implementation. This is not the case for Commercial-Off-the-Shelf (COTS) products, and may not be the case for Government-Off-the-Shelf (GOTS) products. Specifically:

- The COTS vendor usually does not publicize the complete product specification or even a complete list of functions;
- Source code is almost never available;
- The vendor does not warrant the product for satisfactory operation, regardless of claims made in the documentation or promotional literature; and
- The user/evaluator does not own the product, but is licensed to use the product under specific conditions (which may not include reverse engineering to recover design or functional information).

It is also the case that the number of COTS/GOTS products that must be evaluated, particularly in a system-of-systems and open architecture environment may be so large that exhaustive evaluation is not possible with the resources available.

The challenge faced by the Y2K compliance evaluator is to overcome the lack of internal product information and develop some measure of compliance for all products.

Evaluating COTS/GOTS Without a Specification

Since it is given that there is no available specification for a COTS product (and possibly not for GOTS), the evaluator is faced with the problem of accumulating product knowledge from other sources. There are a number of ways to do this:

- What user documentation does exist often contains a number of clues about date processing. In particular, the user interface is likely to be well described, and it may only be necessary to look up the word “date” in the index to establish the explicit date functions performed by the product. Even without an index, a cover-to-cover reading (or a search of a scanned document) should reveal the kinds of specific date functions provided by the product.
- User documentation usually does not describe “non-date functions” that are performed or provided in a date-related fashion (the unique naming of temporary files based on the date and time-of-day, for example). In these cases, users with experience with the product and the product vendor can be helpful.
- As a last resort, a generic list of functions performed by various categories of COTS/GOTS products (The appendix on page 18 is an example) can be consulted.

For each of the date-related functions, one or more test cases can be developed.

It is also useful to examine the system function the COTS/GOTS product is providing in the operational environment. For example, a graphic user interface may provide date displays; and a database management system may provide data storage and retrieval based on date or date-time information. By listing the date-related functions that are important in the operational environment, the evaluator may develop test cases that specifically address the mission-related needs of the user.

Evaluating Many Products Quickly

When there are insufficient resources to exhaustively evaluate all the cataloged COTS/GOTS products, the evaluators must establish a prioritization scheme that can identify those products that may pose the greatest risk to the users’ missions. Typical prioritization parameters will include:

- The importance, *I*, of the product to a specific mission. This can be estimated from the functions the product performs explicitly or from the mission area that it supports. Products that support critical mission areas potentially pose greater risk than those in support areas. *I* can be assigned any convenient numerical range; typical ranges are $0 \leq I \leq 1.0$ and $0 \leq I \leq 10$.
- The pervasiveness, *P*, of the product in the user’s environment. All other things being equal, a product that is installed in large numbers poses more of a risk to the users missions than a product that exists as an isolated instance. *P* can be

expressed as the absolute number or copies of the product that are present, or as a fraction of the total number of copies of all products.

- The compliance level, Y , of the product as determined by an independent source or as claimed by the product vendor. Y ranges from 0.0 (if the product is known not to be Y2K compliant) to 1.0 if compliance is definitely known. The level of compliance of previous releases of the product may be considered in assigning a value to Y .

The Y2K evaluation priority, P , for a product is then

$$P = I \times P \times (1 - Y).$$

Evaluation begins with the products of highest priority and proceeds downward until all products have been evaluated or until resources have been exhausted.

Evaluators should be alert to priorities derived from a compliance level claimed by a vendor. In these cases, some further investigation will be necessary to validate the general quality of the vendor's claims and the value of Y could be adjusted downward if necessary.

Basic Test Cases

For all products that process dates, a set of basic date transition tests cases has been developed. These are shown in Table 1.

For these, or any other selected dates, it *may* not be necessary to execute a complete scenario for every transition. To determine this, the evaluator should examine the ways in which the system under consideration can fail as a result of a critical date transition:

- Crash;
- Immediate or short-term data corruption; or
- Long-term or latent data corruption.

A crash occurs when the software (developed application, infrastructure, or a combination) completely ceases to operate. In the Y2K context, this is most likely to happen at the Y2K transition (midnight, 31 December 1999) because of the inability of one or more components to properly handle the new century. It is, of course, possible, but less likely, that a crash could occur as a result of the leap year transition or the transition to the year 2001. These latter transitions are more liable to cause data corruption or some sort.

Data corruption occurs when a product operates but fails to perform correctly one or more calculations required of it. In the case of immediate or short-term data corruption, the results of the incorrect calculation (failing, say, to recognize Y2K as leap year and therefore computing the number of days between 28 February 2000 and 1 March 2000 to be 1) are visible soon after the error occurs and generally are of a transient and local nature (a bad display, for example). Long-term or latent data corruption, on the other hand, occurs when the results of an erroneous calculation are not detected quickly (often because they are not displayed or reported) and enter into the “database” where their effect on other systems (including those belonging to other organizations) may not be apparent for some considerable period of time (conceivably years). Incorrect logistics or maintenance records are a possible example of long-term or latent data corruption.

Table 1—Basic Test Cases

- **Test the setting and display of dates, including, as appropriate,**
 - **1900/2/29 — should fail — not a leap**
 - **1996/2/29 — to succeed — 1996 is a leap**
 - **2000/2/29 — to succeed — 2000 is a leap**
 - **00/01/01 — to display an unambiguous 4-digit year the value of which is one of either 1900/01/01 or 2000/01/01,**
 - **1999/12/31 —to be able to distinguish between regular of-year 1999 date and an invalid date or example of a never-expiring date**
- **Test the processing of time-data with different data and**
 - **Use the current system clock then test with data dates before and after**
 - **Set the system clock year 2000, say test with data containing before 2000/01/01 and 2000/01/0**
 - **Set the system clock 2000/01/01 and test with containing dates 2000/01/01 and after**

Next the severity of the various failures should be estimated. Candidate categories of severity are:

- Catastrophic;
- Serious;
- Inconvenient;
- Other

What constitutes each of these levels is, to a degree, a function of the nature of the software's purpose and functionality. However, some definitions can be attempted. A catastrophic failure may result in fatalities, serious economic loss, or, in Government terms, complete mission failure. A serious failure will result in loss of data, and may require considerable time for recovery, with the attendant additional loss. A inconvenient failure has minimal or no operational consequences.

As a result of a Y2K failure modes analysis, the test dates in

Table 1, and any others identified by the evaluator, that could result in some form of catastrophic failure should be selected as candidates for full testing. Those dates that could result in a failure of a lesser nature can be given a lower priority and the detail associated with the tests run against them can be adjusted according to the degree of risk associated with the possible failures.

Automating Evaluation

Confidence in a system or product is attained by observing correct operation over an extended period of time. Also, the number of variations of date-related inputs and outputs that many systems must process calls for a large number of test cases to ensure that as many real-world possibilities as possible are covered. For both long-term and high volume situations, the prudent tester will consider automating the testing process.

A number of tools exist to assist in the identification of date processing code and the development of test cases, but here we look for ways to automate the execution of the tests themselves. This means we need a test “harness” that can provide input data automatically to the system under test (the SUT) and that will receive output from that system and record it. Moreover, this harness should be able to compare the recorded output with predetermined responses to determine the correctness of the system’s output.

Automating the input injection and output recording, while a significant task, is generally straight forward. Determining the “right answers” is not so easy. The entity that develops the predetermined or predicted responses is called an *oracle*.

Oracles can be human beings that examine input data and predict responses, or, preferably, they can be automata, that is, tools. If a tester wishes to use an automated oracle, the question arises: where do we get it? There are two possibilities: develop one or use an existing product. If the tester decides to develop an oracle, then he or she has the problem of verifying it.

However, it is likely that an oracle already exists, namely the unmodified (non Y2K compliant) version of the SUT.

The ability to use the system under test as its own oracle arises from the fact that the sequence of days-of-the-week and calendar dates is repetitive. The sequence exactly repeats every 28 years and partially repeats more frequently than that.

The sequence of days-of-the-week and dates for the years 1999/2000 is exactly the same as the sequence for the years 1971/1972. Moreover, the 1999/2000 sequence up to, but not including 29 February is the same as 1993/1994 (1994 was not a leap year, 2000 is).

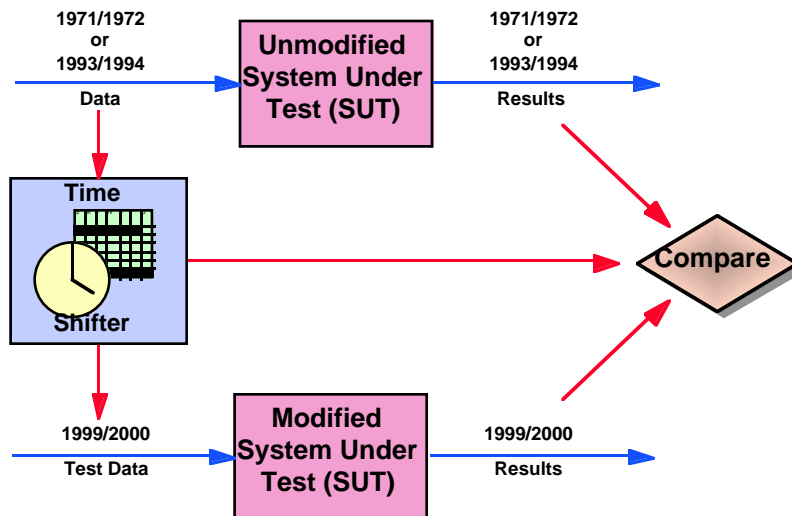


Figure 7—An “Oracle” for Y2K Testing

Depending on convenience and availability, the tester can provide 1971/1972 or 1993/1994 data to the SUT and record the results, these can then be considered “correct” since the SUT would, generally, not be a candidate for Y2K compliance evaluation if its current operation were not satisfactory. These correct results can then be the basis for the oracle’s operation.

The test data for 1971/1972 or 1993/1994 that is to be inserted into the oracle is processed by a “time shifter” tool to convert dates to the 1999/2000 context. The results of the shifting are then injected into the SUT. The output of the SUT is captured and compared to the output captured from the oracle (after time shifting). The two data sets should be identical or should differ only in ways that can be explained and that are derived from the changes that have been made. This process is shown in Figure 7.

Layered Evaluation

It is not always possible or even desirable to evaluate a product for Y2K compliance while it is in its operational configuration (that is, while connected to a complete system or system-of-systems). There are several reasons for this but the primary ones are the complexity of the test and the fact that, at least for new developments, the final system is not available until very late in the development effort when error correction is difficult and expensive.

The best way to evaluate a product or set of products is to use a layered approach. That is, the product is tested as a stand-alone application, then in an isolated system, then finally in a system-of-systems environment.

Evaluating a product as an isolated application has the advantage that scheduling is relatively easy and test cases can be targeted specifically to functions performed by the product. The disadvantage is that the vulnerability of the product to non-Y2K information from another product cannot be completely determined since it is usually impossible to explore a really wide range of erroneous input that an interfacing application or system might provide.

Evaluating a product in a system environment provides some information on the effects it has on external systems. In particular it allows an evaluation of the compatibility of the product’s Y2K approach with those of the other products in the same system. However the functioning of interfaces to remote systems is still a question, and it may be difficult to schedule time on a mission critical system for an extensive test.

Testing a product in a system-of-systems environment, particularly with life-like data, can be the ultimate source of compliance information. But it is difficult to coordinate and usually is expensive since the efforts of the remote systems often must be funded as part of the local testing effort (although in the case of Y2K, where everyone is in the same boat, this may not be true). Also, the functionality of the product being evaluated must be carefully examined. If the SUT provides services that are, to a high degree, internal to a system, the knowledge gained by a system-of-systems evaluation may not add sufficiently to the product evaluation to justify the high cost.

Isolating Systems for Evaluation

When evaluating the operation of a product or system with another system that is not (or is not yet) Y2K compliant, it will be necessary to insert an *intermediary* function (often called a *data bridge*) between the two systems (see Figure 8). This intermediary ensures that the SUT sees data that is Y2K consistent and that the external system sees data in the form it expects.

Depending on the nature of the test, the intermediary may limit its activities to simple format adjustments. If, however the test is intended to simulate operations near or after the Y2K transition, the intermediary will have to adjust actual date values from the external system to provide the SUT with a futuristic environment and will have to alter the output of the SUT so that the external system sees data as it expects it.

In planning a test the evaluators must ensure that the users and administrators of the external system are aware that the SUT and the intermediary are performing an evaluation.

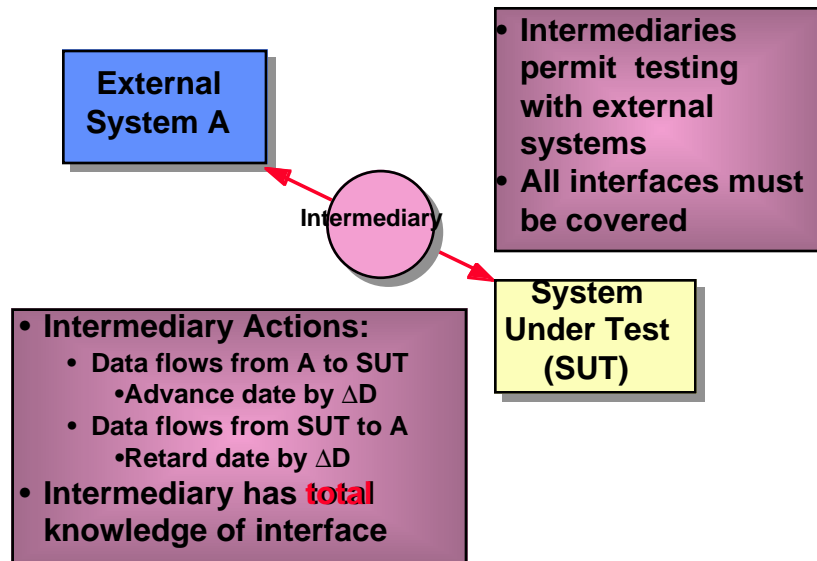


Figure 8—Isolating Systems for Evaluation

A Caution Before Testing

Caution should be exercised when evaluating a product for Y2K compliance when the tests being executed involve simulating the near- or post-transition time frame. Abruptly advancing a system date will cause the product and the underlying infrastructure (in particular, the operating system) to detect a quantum leap in time. Such a leap may cause things to “expire” and in ways that may not be reversible. For example:

- User IDs and access privileges may expire, either because they have been programmed to elapse after a fixed amount of time, or because the system believes that the user(s) have not accessed the system for a long period of time and are, therefore, dormant.
- Passwords may expire since most systems are set-up to require a change in passwords after fixed amounts of time, usually on the order of months. In some cases, setting the date back may undo the problem, in others, it may not.
- Data files and data bases may expire. Many systems will purge files (usually after an automatic archive, but frequently without warning) that have been unused for a set period of time. Abruptly advancing the date may trigger such logic, and the “damage” will likely be done before it is detected.
- System authorizations and protections, like user IDs may expire after a “quantum advance” in system time. Depending upon the mechanism used in purging such authorizations, this effect of date advancing may not be reversible.
- Many licensed products are programmed with “time bomb” logic that cause them to become inoperable if the validity period of the license is exceeded. Once the license has expired, vendor assistance may be required to restore a product to usability.

External Testing Facilities

Most DOD test organizations have elected not to provide Y2K-specific test facilities, choosing, rather, to look to their own Y2K problems. These organizations, such as AF TE and AFOTEC prefer to ensure that their facilities will be operational and supportive of all test and evaluation activities up to and beyond the Y2K transition. These organizations are, however, more than willing to share information about testing and about the Y2K problem itself.

DISA's Joint Interoperability Test Center, JITC, has established an active Y2K support and test program that will advise and assist other organizations on a fee-for-service basis. JITC Y2K contacts are

Year 2000 Manager: Mr. Jack Brandt at brandtj@fhu.disa.mil, (520) 538-5057 (DSN 879), and

Year 2000 Test Director: Mr. Leo Hansen at hansenl@fhu.disa.mil, (520) 538-5053 (DSN 879)

Further, ESC's Command and Control Unified Battlespace Environment (CUBE) is chartered to provide interoperability test facilities for systems that interface to other C² systems. The CUBE, while not providing a Y2K-specific program, will assist program managers and product evaluators in ensuring, to the extent possible, that their products are compatible with the C² warfighting environment.

Summary

A good evaluation program is carefully planned in advance. It begins with a definition of Y2K compliance and uses a compliance checklist to keep itself and others informed of its progress. Further, the good program works continually and uses all information about a system or product, including requirements, design, code, and externally generated information.

Product evaluation is done in layers; first with the product alone, and then in more and more realistic configurations.

For COTS/GOTS evaluations, or any evaluation program where resources are limited, a risk-based strategy should be adapted that permits more thorough evaluations of those products that may have the most impact on mission success.

APPENDIX – Common Date-Dependent Non-Date Functions of COTS/GOTS

Many COTS/GOTS infrastructure products use the date to perform what are essentially non-date functions. Some of the most common ones are listed in Table 2. If these functions are important to the user's operational environment, then test cases must be developed that explore these areas in the near- and post-Y2K time frame. For example, many application-specific computer programs utilize temporary data files. If these programs make use of the operating system's file system for creation and deletion of these files, then it is important to ensure that they continue to operate properly after the year 2000.

Table 2—Non-Date Functions Affected by Dates

Major COTS/GOTS Category	Date Usage
Operating Systems	Names of temporary files Data archiving Time and date values Passwords File management (archiving and purging)
Database Management Systems	Date storage Date and time arithmetic
Mathematical Libraries	Random number generation Date and time arithmetic
Compilers	Time/date-stamp object code
Communications	Date-Time-Group

The list above is, of necessity, incomplete, since the number of ways dates can be used for non-date functions is limited only by the imagination of the product designers.

APPENDIX – The Department of Defense Year 2000 Compliance Checklist

The following pages contain the Department of Defense Year 2000 Checklist. This checklist can also be found on-line at http://www.mitre.org/research/cots/COMPLIANCE_CHECKLIST.html.

DOD YEAR 2000 COMPLIANCE CHECKLIST

(31 Mar 1997)

The purpose of this checklist is to aid system managers in ensuring their systems are compliant for the Year 2000. Make sure the following items are included in your Year 2000 testing and certification process for all of the developed, gratis, licensed, and purchased software, hardware, and firmware used in your system’s operation, development/maintenance, support, and testing activities.

Please respond to each question with the appropriate answer.

System Identification

(An asterisk indicates an optional question)

1. Please provide system information.

a. Name of system

b. Defense Integration Support Tools (DIST)
Number of system

c. Operational date of system (current or a future date)*

d. Planned or actual replacement date of system (retirement or discontinuation qualifies as replacement)*

e. For planned replacements what is the contingency plan and under what conditions will it be invoked?*

f. What are the safety critical portions of the system, if any?*

Year 2000

2. Each system has its own window of time, before and after the present date, in which it functions. Planning and scheduling systems work with dates that are weeks, months, and sometimes years in the future. Likewise, trend analysis systems and billing systems regularly reference dates in the past. For your system, and its window of time, please verify its ability to successfully process data containing dates with no adverse effect on the application's functionality and with no impact on the customer or end user beyond adjustment to approved changes in procedures and data formats.

	VERIFIED	NO	N/A
a. Dates in 20th century (1900s)	_____	_____	_____
b. Dates in 21st century (2000s)	_____	_____	_____
c. Dates across century boundary (mix 1900s and 2000s)	_____	_____	_____
d. Crosses 1999 to 2000 successfully	_____	_____	_____

Other/Indirect Date Usage

3. Have you verified performance (and corrected if necessary):

	VERIFIED	NO	N/A
a. Dates embedded as parts of other fields	_____	_____	_____
b. Dates used as part of a sort key	_____	_____	_____
c. Usage of values in date fields for special purposes that are not dates (e.g. using 9999 or 99 to mean “never expire”)	_____	_____	_____
d. Date dependent activation/deactivation of: passwords, accounts, commercial licenses	_____	_____	_____
e. Date representation in the operating system’s file system (creation dates and modification dates of files and directories)	_____	_____	_____
f. Date dependent audit information	_____	_____	_____
g. Date dependencies in encryption/decryption algorithms	_____	_____	_____
h. Date dependent random number generators	_____	_____	_____
i. Date dependencies in firmware	_____	_____	_____
j. Personal Computer BIOS and RTC does not reset the year to 1980 or 1984 on reboots after 31 December 1999 <i>(corrections by operating system utilities allowed)</i>	_____	_____	_____

Leap Year

4. System accurately recognizes and processes Year 2000 as a leap year.

	VERIFIED	NO	N/A
a. February 29, 2000 is recognized as a valid date	_____	_____	_____
b. Julian date 00060 is recognized as February 29, 2000	_____	_____	_____
c. Julian date 00366 is recognized as December 31, 2000	_____	_____	_____
d. Arithmetic operations recognize Year 2000 has 366 days	_____	_____	_____

Usage of Dates Internally

5. Internal application usage of dates and date fields must be clear and unambiguous in the context of the systems which use them.

	VERIFIED	NO	N/A
a. Display of dates is clear and unambiguous (the ability to correctly determine to which century a date belongs either by explicit display, i.e. 4-digit year, or system or user inference)	_____	_____	_____
b. Printing of dates is clear and unambiguous	_____	_____	_____
c. Input of dates is clear and unambiguous	_____	_____	_____
d. Storage of dates is clear and unambiguous	_____	_____	_____

External System Interfaces

6. External interactions are identified and validated to correctly function for all dates.

	VERIFIED	NO	N/A
<p>a. Interaction between this system and any other external time source, if existing, has been verified for correct operation.</p> <p>For example, the GPS system is sometimes used as a time source. Many GPS receivers cannot correctly deal with the roll-over of the GPS 10-bit epoch counter that will occur at midnight, 21 August 1999. GPS receivers also deal with an 8-bit Almanac Week counter which has a 256 week roll-over span.</p>	_____	_____	_____
<p>b. You and the responsible organization for each interface have negotiated an agreement dealing with Year 2000 issues.</p> <p>For example, is the interface currently Y2K compliant, is it being worked on, does it have an unknown fix date, or will it be fixed by a future date you have mutually agreed on.</p> <p>For each interface that exchanges date data, you and the responsible organizations have discussed and verified that you have implemented consistent Year 2000 corrections that will correctly work for date data passed between your systems.</p>	_____	_____	_____

Date Field Type

7. Describe the type of date fields used by the system, in either software or data bases.

	VERIFIED	NO	N/A
a. Does the system use 4 digit year data fields?	_____	_____	
b. Does the system use 2 digit year data fields?	_____	_____	
c. If 2 digit, does the system use a century logic technique to correctly infer the century?	_____	_____	_____
d. At what date will the century logic fix fail?	_____	_____	_____
	YES	NO	
e. Are there any internal data types for dates?	_____	_____	

If yes to e, what is the range of dates that the date field can represent?

Minimum Date _____

Maximum Date _____

Year 2000 Testing Information

8. Optional: Please provide the following information with regard to testing the application for Year 2000 compliance:

Narrative Answer

- a. Testing Organization _____
 - b. Name of Test Team Chief _____
 - c. Date that Year 2000 compliance testing was completed _____
 - d. How was Year 2000 compliance determined? (certified by vendor or contractor, tested in-house, inspected but not tested, etc.) _____
- | | YES | NO |
|---|-------|-------|
| e. Are the test data sets available for regression testing on the next version release for questions 2, 3, 4, 5, 6, 7d, and 7e? | _____ | _____ |
| f. Are the detailed test results and reports available for review and audit for questions 2, 3, 4, 5, 6, 7d, and 7e? | _____ | _____ |
| g. Do you follow a defined process for tracking the status of all Year 2000 problems reported, changes made, testing, compliance, and return to production? | _____ | _____ |

COTS/GOTS Components

9. Optional: Please provide the following information with regard to COTS/GOTS components.

	YES	NO	N/A
a. Does the system use COTS/GOTS application packages and/or infrastructure components?	_____	_____	_____
b. If yes, have those items been verified to be Year 2000 compliant?	_____	_____	_____

Narrative Answer

c. How was Year 2000 compliance determined? (certified by vendor or contractor, tested in-house, etc.)

Certification Levels

Certification levels are defined below. Yes, verified and N/A are considered positive responses. No is considered a negative response.

LEVEL

- 0 System retired or replaced
- 1a Full independent testing completed with either:
 - All questions have positive responses except possibly 7b and e
- 1b Full independent testing completed with either:
 - All questions have positive responses except possibly 7a and e
- 2a Independent audit of system and existing testing completed with either:
 - All questions have positive responses except possibly 7b and e
- 2b Independent audit of system and existing testing completed with either:
 - All questions have positive responses except possibly 7a and e
- 3 Self-certification
CAUTION: Self-certification assumes a higher risk level of potential failures
- 3a Self-certification with full use of 4 digit century date fields
 - All questions have positive responses except possibly 7b and e
- 3b Self-certification indicates risk due to use of 2 digit century fields
 - All questions have positive responses except possibly 7a and e
- 3c Self-certification indicates risk due to ambiguous usage of dates
 - Question 5-a,b,c or d have negative responses.
- 3d Self-certification indicates potential problems (System needs additional work before Year 2000 processing can be assured with any level of reliability)
 - Question 2-a,b,c or d have negative responses, or
 - Question 3-a,b,c,d,e,f,g,h,i or j have negative responses, or
 - Question 4-a,b,c or d have negative responses, or
 - Question 5-a,b,c or d have negative responses, or
 - Question 6-a or b have negative responses, or
 - Question 9-b has a negative response.
- 4 Not certified or not certified yet.

It would be advisable but not required for the system/program/project manager to have the responsible programmer(s) fill out a similar checklist covering the software they are responsible for before completing this checklist for the overall application.

LEVEL OF CERTIFICATION FOR THIS DATA SYSTEM: (*Circle only one*)

0 1a 1b 2a 2b 3a 3b 3c 3d 4

I certify that the information provided above is true and correct to the best of my knowledge and belief:

ADDITIONAL
COMMENTS: _____

System Manager

Date

I certify that the information provided above is true and correct to the best of my knowledge and belief:

ADDITIONAL
COMMENTS: _____

System Customer

Date