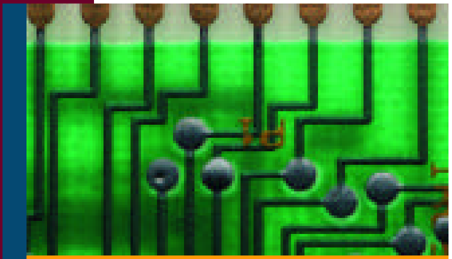

BY GERHARD ADAM

Year 2000: Putting Testing Into Perspective

With the Year 2000 clock ticking, it's important to know what needs to be tested and not waste time testing extraneous dates.

AS work progresses on Y2K, it seems that there is an ever-growing list of conditions that are being considered and tested. There have been so many Y2K dates identified as being potentially volatile that testing itself will probably take most of the next millennium to complete anyway. So how important are all these concerns?

Let's focus on the problem. First, it's not about the Year 2000. Rather, the problem is about two-digit years being employed by systems that perform calculations and comparisons on these values. If only two-digit years are being used, then the Year 2000 is the first year in which error conditions will manifest themselves — the nature of the errors being directly related to the fact that 00, following 99, is not a correct numeric sequence. So what exactly needs to be tested?

SIGNIFICANT DATES

There are an increasing number of conditions identified as being critical to achieving Y2K compliance that are in fact largely irrelevant. Just to pick a particular example, the curious notion that 9/9/99 (September 9, 1999) is a date that could imply some sort of programmatic "high value" or something of special significance is being published everywhere. The reality is considerably less dramatic. This is an insignificant date and doesn't require any special consideration because the notion that all 9's will be stored isn't true. In all date fields, there must be an allowance for two digits for the day and two digits for the month (irrespective of year fields). This simple fact already guarantees that the stored values would actually be 090999 and not 999999. It would be a curious program that used such a value to signify anything.

Similarly, dates are being mentioned for 1/3/2000 (as, arguably, the first business day), 1/31/2000 (as the first month-end), etc. etc. In fact, since the problem exists only because of our use of two-digit years, then when the basic condition of handling the year has been met, the others logically follow and don't require any special consideration.

In cases where program logic has been modified to achieve compliance (rather than expanding the date fields), the following rule should be employed: Test what was changed.

In cases where program logic has been modified to achieve compliance (rather than expanding the date fields), the following rule should be employed: Test what was changed.

This is the only real criterion necessary to determine the range and scope of Y2K testing. Obviously, the more exotic a programming solution, the more complex the testing. It's important to recognize that the issue here is not which dates are significant, but rather, what is the program logic that needs to be exercised?

Notice also, that if the date fields are expanded so that the only program change is in file definitions,there is no logic affected by this change. This has the benefit of allowing testing to occur with existing data,

Figure 1: Two-Digit Year

```
COBOL Program.

Working Storage Section.

77 AGE                  PIC 9(2).
01 .....
       05 CURR-YEAR     PIC 9(2).
       05 CURR-MON      PIC 9(2).
       05 CURR-DAY      PIC 9(2).
01 .....
       05 BIRTH-YEAR    PIC 9(2).
       05 BIRTH-MON     PIC 9(2).
       05 BIRTH-DAY     PIC 9(2).

PROCEDURE DIVISION.

Subtract BIRTH-YEAR from CURR-YEAR giving AGE.
```

Figure 2: Four-Digit Year

```
COBOL Program.

Working Storage Section.

77 AGE                  PIC 9(2).
01 .....
       05 CURR-YEAR     PIC 9(4).
       05 CURR-MON      PIC 9(2).
       05 CURR-DAY      PIC 9(2).
01 ......
       05 BIRTH-YEAR    PIC 9(4).
       05 BIRTH-MON     PIC 9(2).
       05 BIRTH-DAY     PIC 9(2).

PROCEDURE DIVISION.

Subtract BIRTH-YEAR from CURR-YEAR giving AGE.
```

Figure 3: Windowing Technique

```
COBOL Program.

Working Storage Section.

77 AGE                  PIC 9(2).
01 WK-CURR-YR.
   05 WK-CURR-CI        PIC 9(2).
   05 WK-CURR-YEAR      PIC 9(2).
01 WK-BIRTH-YR.
   05 WK-BIRTH-CI       PIC 9(2).
   05 WK-BIRTH-YEAR     PIC 9(2).
01 .....
   05 CURR-YEAR         PIC 9(2).
   05 CURR-MON          PIC 9(2).
   05 CURR-DAY          PIC 9(2).
01 .....
   05 BIRTH-YEAR        PIC 9(2).
   05 BIRTH-MON         PIC 9(2).
   05 BIRTH-DAY         PIC 9(2).

PROCEDURE DIVISION.

If CURR-YEAR < 50 move '20' to WK-CURR-CI else move '19' to WK-CURR-CI.
Move CURR-YEAR to WK-CURR-YEAR.
If BIRTH-YEAR < 50 move '20' to WK-BIRTH-CI else move '19' to WK-BIRTH-CI..
Move BIRTH-YEAR to WK-BIRTH-YEAR.
Subtract WK-BIRTH-YR from WK-CURR-YR giving AGE.
```

since the condition being tested is to ensure that no definitions were missed. The actual application logic is untouched and therefore doesn't require testing.

Let's use a simple example to illustrate this point. By examining the two programs shown in Figures 1 and 2, it's easy to see that nothing has changed within the PROCEDURE DIVISION and therefore there is nothing special that needs to be tested. The operation of the program remains exactly the same as before the change. This means that the only testing needed will be to ensure that changes to the date field lengths operate properly. This does not require any special dates to be considered, but will be just as conclusive with 1998 dates as with any other. This can have significant implications when establishing test scenarios.

There have been so many Y2K dates identified as being potentially volatile that testing itself will probably take most of the next millennium to complete anyway.

Let's look at Figure 3 where a windowing technique is employed. In this case, program logic has been added to make the decision as to how the century is to be interpreted. Changes were made to the working storage areas and names within the program were modified to use these new values. Proper testing of this application will require that Year 2000 dates be employed, since there is program logic that will be explicitly triggered by its use. In this case, the application will require that dates around the selected "window" be employed to exercise the logic to ensure its proper functioning.

Similarly, if other techniques are employed such as encoding, or setting century indicators, those specific techniques will have to be tested based upon the logic introduced into the program.

It's important to notice that by focusing on the specific changes within the application, we have removed much of the uncertainty pertaining to the supposed "significant" dates of the Y2K problem. There are actually only two specific conditions that occur with respect to the Y2K problem: the actual transition to the Year 2000 whereby the use of two digits is a problem and the occurrence of a leap year.

In the event that dates have to be advanced to test application logic, there are two ways in which this must be addressed.

Time Machines

The issue of using "time machines[1]" is based on the idea that testing must occur within a configuration that is using future dates to simulate future conditions. As indicated, such a test environment is not necessary for four-digit dates and is usually not necessary unless specific programming changes require such tests. Examples of such conditions are screen input or current date retrievals. These are the only two situations in which a time machine actually makes a difference.

This environment is also necessary to test systems software and many system management functions since they generally operate from current date. For example, to test security functions or storage management it would be necessary to advance the system clock to see how the system behaves under those conditions.

Note: *Testing systems software by advancing the date should never be done against live data or files. Unpredictable behavior can cause data to be lost or contaminated. It should be tested against test files and DASD volumes just like applications would be tested against test datasets.*

Since most of the dates being operated on occur within data files, a far more serious issue is how to properly "age" dates within files.

Aging Dates

"Aging" dates is the process whereby the dates within files are advanced so that they reflect conditions as they would be in the future. This is required so that testing of an application will simulate operation at the future time being tested.

To properly "age" dates within files requires a very explicit knowledge of the application and the relationships of dates to each other. This is especially important when program logic is to be exercised that is dependent on certain types of interactions.

For example, certain date fields will "age" whereas others will remain constant. Dates of birth, dates of hire, termination, etc., are all dates that typically don't change, so caution must be exercised with respect to "aging" date fields within a file.

Other situations that require consideration include not having the application operate outside of its design range. If we were to take a payroll system with its databases reflecting current dates, while the system date was advanced to the Year 2000, the payroll system would suddenly experience an 18-month gap in its processing. Testing under such circumstances would produce results that are suspect since the application was never designed to operate under those conditions.

HARDWARE

Hardware is probably the most straightforward to test. In all cases, the system

In cases where program logic has been modified to achieve compliance (rather than expanding the date fields), the following rule should be employed: Test what was changed.

Embedded Technology

Embedded Technology

Embedded technology represents all those computer chips that play a role in our daily lives and may be subject to date conditions. This can range from elevators to fax machines to voice mail systems to lab equipment. Each represents a potential for failure and needs to be examined.

When dealing with embedded technology, it is critical that vendor contact be established to determine compliance and exposure information. It is equally important that qualified personnel conduct such tests. In many cases, serious failures or damage can occur to equipment that is improperly modified.

In addressing embedded technology, there are several points that may help in prioritizing concerns:

1. Embedded technology must have a power source to have a potential for Y2K impact.
2. There must be a means whereby a date is displayed or modified. If this doesn't exist, then it is unlikely that the operation of the equipment will be affected in any way by the Y2K problem.
3. Evaluate whether having an improper date actually impacts operation. For example, a microwave oven may have a bad date/time, but this doesn't prevent it from operating.

While these points don't address all of the concerns regarding embedded technology, they represent a starting point for evaluating potential impacts. In many cases, the failure of a date-related function will not necessary impact the operation of the equipment in question.

In addition to researching embedded technology, it may be necessary to establish contingency plans to deal with potential failures. In some cases, the contingency plan may be as simple as "turning off" the equipment if it fails. Specific issues should be examined for each type of technology with contingency plans in place should failure occur. In some cases, where ramifications may be quite serious, it may be necessary to explore the possibilities of manual intervention or override. In addition, higher levels of employee awareness may be necessary to ensure that immediate actions are taken in the event that equipment begins to operate out of "character."

1. The points relating to "time machines"may also be applied to date simulation products. While they may be useful in certain applications, it is important to realize that they do not represent a complete solution for testing Y2K date conditions.

clock must be set to December 31, 1999. Choose a time close to midnight and then power off the machine (**Note:** *This is not practical for large mainframe systems that would not require a power-down.*) Restarting the machine after "midnight" will determine whether the system clock has properly advanced into the Year 2000.

If this basic test fails, then you must determine whether replacement or manual intervention is sufficient to address the problem. In most cases,a hardware upgrade may be in order to avoid difficulties with applications retrieving hardware dates.

VENDOR SOFTWARE

Compliance letters are often provided by vendors to indicate that the application will function properly and should be obtained whenever possible. In addition, testing should consist of:

1. Confirm vendor claims of compliance by testing specific operating conditions of the application. If possible, it should be validated that the vendor's technique of compliance operates under general usage.
2. If anomalous behavior is detected, a determination needs to be made as to the impact on the operation of the product. In many cases, the behavior may simply represent a restriction on the portability of the data or require a "bridge" before data can be ported.
3. Identify all date restrictions based on vendor correction technique. This is necessary to provide documentation for future application uses and possible interfaces.

CONCLUSION

It's important to keep a perspective on the Y2K problem and not get caught up in a flurry of activity that doesn't produce the necessary results. With time running short, it is important that the Y2K testing activity is conducted efficiently. Testing requirements are no different for the Y2K than for any other application. It is important that we don't fall into the irrational domain of panic, whereby everything must be tested. This has never been a practical approach and isn't one now. We must test those things that have changed and remain focused on the actual functional issues that are at stake. ts

Gerhard Adam, president of SYSPRO, Inc., has 26 years of experience in large systems computing, specializing in performance and MVS/ESA internals. He has been involved in development that extends from access method interfaces and telecommunications to AFP software.