

Using Test Cycles for Testing Year 2000 Projects

Randall W. Rice
Rice Consulting Services

A major challenge in enterprise-wide system testing is to devise a test process that simulates the operation of the organization over a period of time and that covers the processing of many systems. Enterprise-wide testing is extremely complex, and the need for such testing is particularly evident in most Year 2000 testing efforts. However, traditional testing processes often test software and systems at snapshots of time as opposed to testing transactions through specific checkpoints or cycles. The test-cycle approach in this article describes how to construct a set of tests that are controllable, repeatable, and measurable across any given span of time.

A variety of testing techniques are available that can easily be adapted and applied to Year 2000 (Y2K) projects. One technique that greatly helps plan, coordinate, document, and track testing is the test-cycle technique. In this technique, testing is organized and performed in cycles that can be defined to simulate specific dates. The great value in this approach is its application to large-scale enterprise systems. This kind of “end-to-end” test is especially critical in Y2K projects because of the need for testing interfaces between many different systems, both internal and external to the enterprise. This article presents an overview of the test-cycle concept, the benefits of using test cycles, and an example of how test cycles can facilitate Y2K testing.

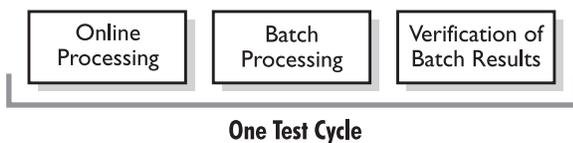
The examples presented in this article may appear simple because they are presented for the sake of illustration. In the real-world application of the test-cycle approach, the scope can be much larger but these techniques may still be used to manage the complexity of large-scale systems or user acceptance testing.

What Is the Test-Cycle Concept?

First, a test cycle is any defined period of testing. A test cycle could simulate a day, a week, a month, or no period. The ability to simulate a given period, however, is what makes test cycles an ideal technique for date-sensitive testing.

Exactly what happens during a test cycle depends on the technology involved. For example, in a traditional legacy mainframe environment, a test cycle usually consists of three parts: on-line data entry, batch processing, and the verification of batch results (Figure 1). In an environment that does not contain batch processing, the test cycle consists of interactive processing only. In a batch-only environment, a test cycle would consist of batch processing followed by the verification of batch results.

Figure 1. *Traditional test cycle.*



ID	Description	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7
		12/15/1999	12/31/1999	1/1/2000	1/3/2000	1/15/2000	2/28/2000	2/29/2000

Figure 2. *Matrix headings with test-cycle dates.*

For each test cycle, a simulated processing period can be defined. That is why test cycles are an ideal way to plan and organize Y2K testing. One test cycle can be set for 12/15/1999, another cycle defined as 12/31/1999, another at 1/1/2000 and so on. The test environment date for each test cycle will need to be set using a date simulation tool.

The number of test cycles required for a test will depend on the amount of simulated time to be spanned during the test. For example, if you are merely testing the date rollover, you will only need a few cycles—probably 12/31/1999, 1/1/2000, and 1/3/2000 (the first Monday in 2000). However, if you are going to perform a more complete Y2K compliance test, you will need to define test cycles that allow a longer span of testing. For example, if you are testing a 30-day cancellation period across the century rollover, you might have one cycle defined as 12/15/1999 and another at 1/15/2000. You would also want other test cycles defined at 2/28/2000 and 2/29/2000 to test leap year processing. With the test-cycle approach and a date simulation tool and a data aging tool, you can define cycles as far in the future as you like. So, for testing leap year processing, you could also have cycles for 2/28/2004 and 2/29/2004.

Within each test cycle, one or more tests are defined to be performed. In some test cycles, it may be desirable to define no tests, depending on the cases being tested. The tests may be defined using test scripts or test cases.

The Process of Defining and Using Test Cycles

Now that test-cycle concepts have been discussed, let us look at the details of planning a Y2K test using test cycles.

Step 1 – Make Sure You Have the Right Tools

You will need a date simulation tool to easily change your test environment dates. You will also need a data aging tool to

ID	Description	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7
		12/15/1999	12/31/1999	1/1/2000	1/3/2000	1/15/2000	2/28/2000	2/29/2000
1	DED = \$500							
2	DED = \$1,000							

Figure 3. Test cycle matrix with cases.

advance the dates in the test data and keep the relationships synchronized.

Step 2 – Define the Dates You Will Need to Simulate

These simulated system dates will depend upon the extent of your testing—namely, the levels of Y2K compliance you need to validate. There are four basic categories of Y2K compliance to consider:

- No value for the current date will cause interruption in operation. No matter what the system date is, the system will work correctly.
- Date-based functionality must behave consistently for dates prior to, during, and after 2000. All functions using dates as a basis should be correct. This includes calculations in the 19th, 20th, and 21st centuries, and calculations that span those centuries.
- In all interfaces and data storage, the century in any date must be specified either explicitly or by unambiguous algorithms. Either the century must be explicitly shown in the date, e.g., as a four-position field or by using a century indicator, or by using a logic routine to interpret the date based on a window of time or some other method.
- The year 2000 must be recognized as a leap year. If your system processes data from early in the 20th century, you need to be able to distinguish 1900 from 2000 for leap year purposes.

The dates that many people are using as system test dates at a minimum are

- 1/1/1999
- 9/9/1999
- 12/31/1999
- 1/1/2000
- 1/3/2000
- 1/4/2000
- 2/28/2000
- 2/29/2000

Your specific system dates will depend on your applications, business, and technology needs.

Figure 4. Test cycle with test ID numbers.

ID	Description	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6	Cycle 7
		12/15/1999	12/31/1999	1/1/2000	1/3/2000	1/15/2000	2/28/2000	2/29/2000
1	DED = \$500	A001	A002	A003		A004	A005	A005
2	DED = \$1,000	A001	A002	A003		A004	A005	A005

Step 3 – Build a Test-Cycle Matrix

Spreadsheets are great tools for this. You need to leave at least the first two columns blank for the test case identification (ID) and description, then define the test cycles along the top of the spreadsheet (Figure 2).

Step 4 – Define the Test Cases or Business Cases to Be Placed on the Matrix

Test cases and business cases are those entities you intend to test. These cases will go through one or more cycles of testing and will execute several test scripts or test scenarios. This approach to testing is what gives the test cycle concept so much power. You get to simulate not only the effect of the century rollover but also how people and things are processed through your systems from beginning to end. This is in contrast to merely testing one program at a time in a stand-alone fashion.

Figure 5. Sample test script.

Step	Program ID	Action	Expected Result	Observed Result	Pass/Fail
1	ACB001	Enter policyholder number and press <ENTER>.	Policyholder information displayed correctly.		
2	ACB001	With policyholder information displayed, press <F5>.	Control is transferred to billing screen (ACB002). Billing information correct for policyholder.		
3	ACB002	Press <F10>.	Exit to main menu.		

Some examples of test and business cases would be a policyholder, a customer, a patient, or a taxpayer. Each of these entities would then have attributes that would make it unique. For example, if you are testing policyholders, you might have one policyholder with a deductible of \$500 and another with a \$1,000 deductible (Figure 3). The number of test and business cases you include will depend on the level of test coverage you need relative to the risk involved.

Step 5 – Define the Test Order for Each Test or Business Case and Place in Correct Spreadsheet Cell

Each cell can contain a reference to a test or tests that are to be performed for a particular test and business case in a particular test cycle (Figure 4). You might decide to skip a cycle or two for some cases and double up or have several tests in other cycles. Once again, this is an example of how test cycles help you simulate the real world. Just like your live production databases were not instantly created in your business, the test data entered into the system cycle by cycle will continuously build. Keep in mind, however, that every test and business case added to the test will be one more item to maintain throughout the test.

Step 6 – Define the Tests in Detail

For every test indicated on the test-cycle matrix, a detailed description of the test will be needed for documentation both before and after the test. The details should include controls (such as when the test will start and stop), input, expected output, and the test procedure to be followed. An ideal way to

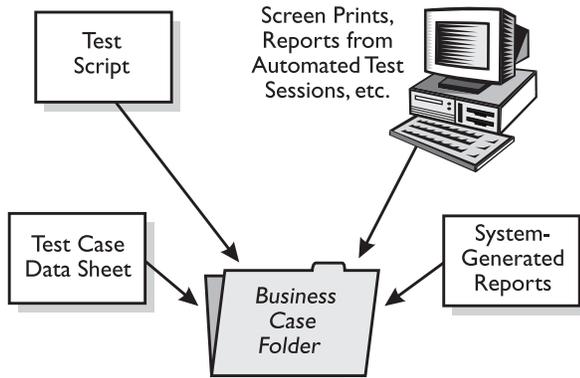


Figure 6. Business case folder.

document these aspects of a test for interactive software is to use a test script (Figure 5). You must determine how much detail is reasonable, given the amount of time you have left for testing and the relative business and technical risk.

Step 7 – Put It All Together

After using this method for many years, I have developed what seems to be a fairly smooth procedure to organize a major test based on the test-cycle concept. Although you can certainly automate testing using test cycles, many organizations do not have test automation tools in place and do not have the time to integrate a tool before starting Y2K testing. In addition, my surveys show that although 80 percent of the organizations I surveyed own an automated test tool, only about 25 percent of those organizations use the tool. For these reasons, the manual application of the test execution process will be shown.

First, you will need a manila folder for each test and business case you have defined. There will be a folder for each row on the matrix (spreadsheet). Label each folder with a business case ID number. This should also correspond to the ID on the matrix. Next, place everything you will need for the business case in the folder. This will include test data and test scripts or test procedures (Figure 6).

To simplify things and to find the right test information quickly, place a cover sheet (Figure 7) on the outside of the folder. The cover sheet shows the test cycles, the test scripts and the procedures performed in each test cycle, and a sign-off column to be initialed by the person who tests the business case.

The final piece is to get as many cardboard bankers' boxes as you have test cycles. If you have only a few folders per cycle,

Figure 7. Folder cover sheet.

Cycle	Test Scripts	Tested	Verified
1	A001		
2	A002		
3	A003		
4			

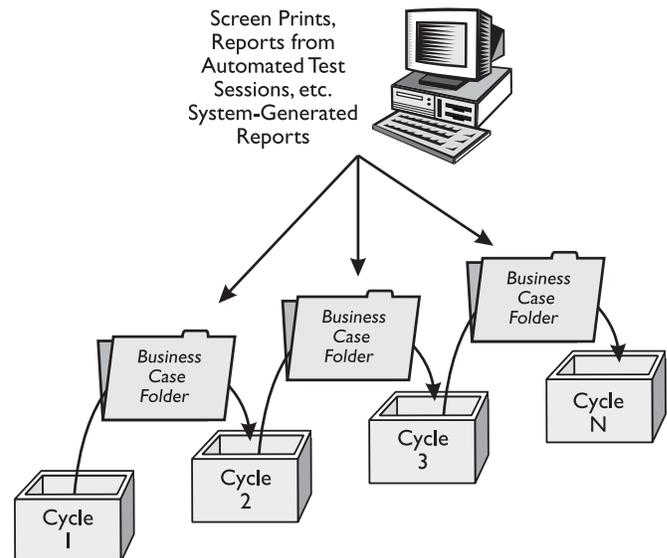
you can get by with using dividers in one or two boxes. You will need a way to start out the test with each set of folders separated by test cycle. Place the folders in the boxes by test cycle and in business case ID order. Each test cycle results in a new collection of these types of folders.

Step 8 – Execute the Test

Start the test by setting the system date with the date simulator to the first test-cycle date. If a bed of test data will be used from the start, make sure the dates in the test data are correct.

Starting with the folders in the first cycle box, perform the tests in each folder for Cycle 1 only. During the test, you might create documentation you would like to save, such as screen prints or reports. These can be placed in the folder, unless the volume is large. In this regard, the test is self-documenting. When the test is complete, initial the folder in the "tested" and "verified" columns on the cover sheet, and place

Figure 8. Test execution process using cycles.



it in the next cycle in which it will be used. If batch processing is part of the test cycle or test procedure, the folder will go back into the same test-cycle division from which it was retrieved. After batch processing is complete, the folder can be pulled, evaluated, and moved on to the next cycle division in which it will be used (Figure 8).

This process continues until the folder is finished and placed in a "done" box. Eventually, all of the business case folders will be filed in the done box in business case ID order. A year or two from now, if anyone needs to know what was tested, it is a simple matter to locate and retrieve the test documentation.

Step 9 – Evaluate and Track the Test

As the test is performed, you will evaluate the results and determine if the test passed or failed in that particular cycle. There are two effective and easy ways to keep track of test progress

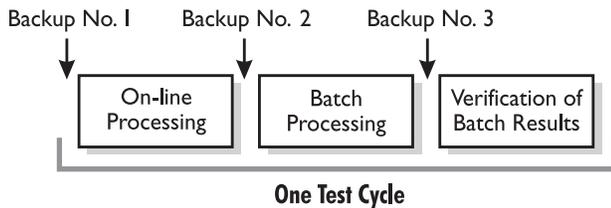


Figure 9. Backups performed in test cycle.

manually. One way is to use the outside cover sheet of the folder to indicate pass or fail. The other is to highlight each cell in the matrix as the test is completed and passed. It is good to use both methods.

The Key Benefits of Using Test Cycles

Although designing test cycles and business cases is extra work, there are some excellent benefits you achieve with no other test method that are especially important for Y2K testing.

- **The ability to simulate a business case from point A to point Z in your processing.** Most other test methods focus on one process or software module at a time, but never have a way to effectively string them together for end-to-end testing of a system or systems.
- **The ability to plan and coordinate the march of time for a test.** For Y2K testing, the tester knows that time must be advanced, but the problem is how to maintain synchronization among the test data, test environment, and test cases. The test-cycle concept allows you to do this with ease.
- **A safety net in case the test environment gets corrupted.** It is common in testing for the test to destroy data or update data files with incorrect information. It also is not uncommon for other people to delete or to restore over test files. The common response to this situation is to restore from the last backup, but how do you know what was tested since the last backup? In most test processes you do not know exactly what was done, but with test cycles, you *do* know. The backup process is fairly straightforward. You take image backups of the test environment before and after on-line input. If batch processing is part of your test, the backup taken after on-line processing will also suffice for the batch backup (Figure 9). These backups should be taken during each test cycle.

Conclusion

In testing, the confidence level of the test depends on the rigor and coverage of the test. The rigor and coverage of the test depends on the relative risk, both business and technical. While some might look at the work involved in planning a test using test cycles as being excessive, others will testify that this kind of effort is required on some projects and systems to validate their operation through multiple simulated dates. The extent of test planning and execution always depends on the scope of coverage and risk. The question is, are you willing to bet your business or systems operation on anything less than the right test method for the job? ♦

About the Author



Randall W. Rice is president of Rice Consulting Services, Inc. and has over 20 years experience building and testing large-scale information systems. He is a certified quality analyst and certified software test engineer specializing in systems testing and the testing of Y2K projects. He is the author of *The Year 2000*

Testing Handbook, creator of the "Testing the Year 2000" workshop, and co-author of *The Top Ten Challenges of Software Testing*. He also is chairman of the Quality Assurance Institute's annual International Software Testing Conference. He has worked with corporations and government agencies worldwide on Y2K testing issues.

Rice Consulting Services
P.O. Box 891284
Oklahoma City, OK 73189
Voice: 405-692-7331
Fax: 405-692-7570
E-mail: rcs@telepath.com
Internet: <http://www.riceconsulting.com>

Coming Events

Call for Papers: Software Engineering Laboratory
Software Engineering Workshop

Dates: Dec. 2-3, 1998

Location: Goddard Space Flight Center, Md.

Topics: Software benchmarks, technologies, environments, standards, requirements capture and validation approaches, methods for safety-critical systems, reuse, COTS-based development (emphasis on process experiences, not products), automatic code generation, process improvement, and measures.

Abstracts (3-5 pages) should be directed to
SEB Abstracts Coordinator
Code 581

NASA/Goddard Space Flight Center
Greenbelt, MD 20771

E-mail (ASCII text only): Jackie Boger,
jboger@cscmail.csc.com

Deadline for receipt of abstracts: Sept. 14, 1998

Internet: <http://fdd.gsfc.gov/seltext.html>.

Camden Technology Conference: The
Transformation of Learning

Dates: Oct. 23-25, 1998

Location: Camden, Maine

Hosts: Bob Metcalfe, Tom DeMarco, and John Sculley.

Subject: The event will gather a faculty of experts from business, technology, government, and academia who will play a major role in shaping the learning methods and technologies of the coming century. Speakers include Alan Kay, Brenda Laurel, Seymour Papert, and Roger Schank.

Contact: 877-223-9752

Internet: <http://www.camcon.org>