Year 2000 Automated Testing
A Summary

Don Estes
2000 Technologies Corporation

For many organizations, it is neither practically nor financially feasible to test all year 2000 program fixes to a level of accuracy that approaches 100 percent. However, there are guidelines to determine whether a system, when it is fielded, is likely to have a containable level of faults. Several automated testing techniques can help attain this level of assurance while helping work within time and financial constraints. This is a summary of a much longer working paper that can be found in the May 1998 Internet version of CROSSTALK *at* http://www.stsc.hill.af.mil/CrossTalk/crostalk.html.

Testing applications for year 2000 (Y2K) compliance is the equivalent of seeking a safe harbor for an anticipated storm. The metaphorical storm will be the organizational and economic disruption that could result from the failure of critical computer systems to function correctly as a result of the date change from 1999 to 2000. The safe harbor thus sought is to ensure, in advance, that the computer systems within the control of the organization will function correctly and will therefore ensure the business against loss.

The complete version of this article details the steps that must be taken to enter that safe harbor or, if this is impractical or inadvisibly expensive to reach, to arrive as close as possible with available resources. However, the scope of the complete article is limited to testing internal application programs. Application systems sited within the organization, if found to be at fault, can be repaired either with in-house or contractor staff or with staff of the licensing vendor. Application systems sited outside the organization that interface to internal application systems must be examined to ensure the data they send and receive remain compatible and that appropriate actions be taken with the correspondent party if incompatibilities

are found and not fixed. Nonapplication systems, if found to be at fault, can in most cases only be replaced.

The primary issue of application testing is that it is not mathematically feasible to test all programs to a level of 100 percent certainty. It may be possible to get close enough to 100 percent to constitute no practical difference—or the scale of the task, the elapsed time required, or associated costs may be so great that significantly less than 100 percent may be the best that can be realistically achieved. Thus, there is the preferred case of risk minimization, which is close to 100 percent, vs. the alternative case of risk optimization, which tries to minimize the risk to the business while accepting that some level of risk will be unavoidable.

Ultimately, the problem can be reduced to a risk vs. cost trade-off. Widely quoted statistics put the cost of testing for a Y2K project at 40 percent to 60 percent of the total cost of the project or roughly a 2-to-1 or 3-to-1 ratio over the cost of renovation. Low accuracy requirement applications, such as noncritical government service applications, may require less than a 1-to-1 ratio of cost of testing to cost of renovation. Conversely, where extremely high accuracy is required, as is frequently the case in the financial industry, the cost can exceed 80 percent of the total cost or more than a 5-to-1 ratio of the cost of renovation. The higher the accuracy required and achieved, the lower the resulting risk of business disruption but

the higher the cost. It has been frequently observed that Y2K testing projects lack sufficient resources for testing to even a modest level of accuracy, and reaching the business case level of accuracy may require substantial increases in resources.

Testing is required to ensure that Y2K compliance modifications made to programs do not introduce new problems and to assure that the programs will continue to operate correctly as the data they process begins to include dates in the 2000s as well as in the 1900s. However, what will be the consequence of inadequate testing? For mainframe and client-server systems, undetected residual program faults will show up in one of two ways:

- Outright program failure, forcing a halt to at least some part of the application processing until the program can be repaired.
- Data corruption, which will force the application completely off-line until the program problem causing the corruption is traced, repaired, and tested and the data errors are repaired.

It is important to keep in mind that any significant data processing installation has some level of faults, as any user of desktop software is reminded daily. However, most faults are too trivial to worry about. Provided there is sufficient staff to cope with nontrivial problems and there is contingency backup for rare crisis situations, it is a *containable* level of faults.

The difference in a Y2K application failure situation is a matter of degree, not of kind. The level of daily faults will reach a point that will overwhelm the support staff; the contingency backup support, which is designed for isolated crises, will also be overwhelmed by simultaneous crisis calls from too many sites. The faults will come in waves as critical dates are reached for each application, and the faults will build to a peak around the end of 1999 and the beginning of 2000. Faults will start to recede after March 1, 2000, although new failures will continue for some time. We are already seeing a few cases of significant Y2K faults, although so far none have been overwhelming.

If a testing project fails to complete full testing, it does not *necessarily* follow that the renovated application will fail in production. In some cases, the level of undetected faults will be containable in practice. In other cases, undetected faults will not be containable, and damage to the business will result. The business purpose behind

significant testing projects is to take chance out of the equation and to provide an insurance policy against damage. In this sense, the cost of the testing project can be considered the premium on an insurance policy.

The complete version of this article details what is required to achieve risk minimization using conventional testing methods, how to proceed in a risk and cost optimization testing project using conventional testing, and a discussion of some innovative technical approaches to introduce economies of scale by automating the process of testing. Where applicable, automated testing can allow a testing project to move significantly closer to the risk minimization model within the limits of what is practical and affordable. ◆

About the Author

Don Estes is chief technology officer for 2000 Technologies Corporation, for whom he has designed and implemented both a data encapsulation and an automated testing system. He also works closely with vendors of limited window-

ing, program encapsulation, and object code remediation systems. He has been involved with COBOL and database applications for 25 years and database and mainframe performance tuning for 10 years. For the last seven years, he has helped design and execute projects for the mass modification of large bodies of source code, primarily for platform migration, using state-of-the-art automated source language transformation technologies and automated testing methods. He is a regular contributor to Peter de Jager's Year 2000 mail list, where he is known for his contributions relating to Y2K rapid compliance strategies and automated testing. Estes is a graduate of Massachusetts Institute of Technology in physics, with a postgraduate degree from the University of Texas in educational psychology.

2000 Technologies Corporation
114 Waltham Street, Suite 19
Lexington, MA 02173
Voice: 781-860-5277, 800-756-8046
E-mail: info@2000technologies.com

inspections unless they start seeing value for the dollars spent. They need to see the business payback in quantitative terms. The business value analysis should compare the cost (hours per major defect) for defects found in inspections with the cost for each test activity. If you have not been collecting this data for the test activities, you may need to have developers and testers estimate the number of hours they believe it takes to find defects during each test activity. ◆

About the Author

John T. Harding is one of the founding partners of Software Technology Transition, which provides training and implementation in the Software Engineering Institute (SEI) Capability Maturity Model (CMM) and CMM-Based Appraisal for Internal Process Improvement method and in software inspections, metrics, and project management. Other work includes the International Organization for Standardization (ISO) gap analysis, helping organizations develop business and software baselines, and action planning for software process improvement. He was a visiting scientist at the SEI, was the metrics mission manager for Groupe Bull, and held various technical and managerial positions in software development with IBM and the Bank of Boston. He has a master's degree in business administration from Boston University and a bachelor's degree from Rensselaer Polytechnic Institute (RPI) and is a member of the

Association for Computing Machinery and the Institute of Electrical and Electronics Engineers.

Software Technology Transition
60 Elm Street
Andover, MA 01810
Voice and fax: 978-475-5432
E-mail: johntharding@compuserve.com

Recommended Reading
1. Ebenau, R.G. and S.H. Strauss, *Software Inspection Process*, McGraw-Hill, New York, 1994.
2. Gilb, T. and Dorothy Graham, *Software Inspections*, Addison-Wesley, Reading, Mass., 1993.
3. Weller, E.F., "Lessons From Three Years of Inspection Data," *IEEE Software*, September 1993, pp. 38-45.
4. Weller, E.F., "Using Metrics to Manage Software Projects," *IEEE Computer*, September 1994, pp. 27-33.
5. Grady, Robert B. and Deborah L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Englewood Cliffs, N.J., 1987.
6. Grady, Robert B., *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
7. Ishikawa, Kaoru, *Guide to Quality Control*, Asian Productivity Organization, Tokyo, 1976.
8. Burr, Adrian and Mal Owen, *Statistical Methods for Software Quality*, International Thomson Computer Press, London, 1996.