Year 2000 Problem Fixes:
Don't Hold Out for a Silver Bullet

Paul Harames
Software Technology Support Center

Despite overwhelming data regarding the year 2000 (Y2K) problem, there are engineers and laymen who believe the Y2K crises is either overhyped or a sham. Although it is unwise to overreact to the Y2K problem, many in the computer world mistakenly believe that quick solutions and improved technologies will somehow save them from the enormous cost and time pressures associated with Y2K fixes. Although some technologies are producing promising results, no method eliminates the need to properly assess, test, and implement renovated systems. Some systems may not need to be fixed at all, but will still require procedure changes and evaluations to assure Y2K compliance. This article discusses issues associated with the year 2000 and explains why although there may be no true silver bullet, Y2K problems may have a silver lining.

The effects of Y2K problems are not imagined—many organizations are already experiencing them, and the threat of litigation between organizations looms because of Y2K computer problems. Government and industry information technology managers are reacting to the problems they have already experienced, and now have data to project the impact if their systems fail. As a result, many have accelerated their Y2K efforts.

However, there are good and bad ways to respond the Y2K problem, as illustrated by an old story of two friends who seized an opportunity to sell watermelons. Several farmers offered to sell them their bumper crop for the super price of 50 cents per watermelon. So the friends bought a large truck and began to sell watermelons rapidly at 55 cents each. A week later, after selling many truckloads of melons, one of the friends said to the other, "Something has got to change. We're losing money." The other friend replied, "Yes, we are losing money, I've been concerned about that as well. What we need to do is buy a bigger truck."

Such is the Y2K effort: failure to correctly diagnose and treat Y2K problems can lead to disaster. Among the worst responses is to assume that someone has or will discover a simple, cheap solution to the problem, or to miscalculate the severity of the problem. It is important to note that the methods and tools fostered from modern improved processes are providing real solutions for Y2K problems; however, system renovation costs account for only a fraction of total Y2K remediation costs. As a result, even the most efficient fixes are not the panacea they might initially seem to be.

On the other hand, overreaction to the Y2K issue can also lead to wasteful mistakes. In the February 1996 issue of *American Programmer*, Nicholas Zvegintzov warns of the dangers of being scared into jumping on the wrong bandwagon, since some people are more concerned about making money from the Y2K issue than they are with providing reliable data and the fastest, safest solutions possible.

The STSC Y2K team agrees, for the most part, with the foundation for warnings about unsavory or overhyped Y2K assertions. Assertions of a Y2K racket, ruse, or farce have probably been used by Zvegintzov and others to curb an overreaction and to reduce superfluous Y2K activities. The hazard of such language is that it can lead to the opposite effect. Care should be taken that the scope of contingency actions are not too narrow, become inert, or that contingency planning is deferred and not present when needed. While many may have already become, or will become, victims of farce, racket, and ruse elements, many more are in need of much more Y2K help and do not even know it. Far more organizations have victimized themselves by underestimating the costs, risks, and difficulties to solve their Y2K problems.

Worldwide Y2K Costs

The Asia Pacific View [1] estimates that worldwide Y2K fixes will average about $1.10 per line of code, a conservative estimate when compared to others. Table 1 lists the cost breakdown estimates for two studies.

Software Technology Support Center (STSC) evaluations of current Air Force efforts indicate that the cost breakdowns in Table 1 are fundamentally sound. Because "simple" Y2K fix techniques are technically easy to accomplish, they have generally already been anticipated and included in the worldwide Y2K cost estimates. The projected $600 billion world-wide conversion cost estimate is difficult to understand because many costs are not accounted for.

Suggested (Adapted from The Gartner Group estimates)	
Awareness	6%
Assessment	20%
Renovation	15%
Validation	50%
Implementation	9%
Asia Pacific View	
Awareness	1%
Inventory	1%
Project Scoping	4%
Analysis/Design	20%
Modification	20%
Unit Test	25%
Systems Test	15%
Acceptance Test	5%
Impl. Documentation	9%
Project Mgt. (add 25 percent to total of above effort)	

Table 1. *Average estimated breakdown of Y2K project costs.*

For example, the Air Force Y2K compliance cost estimate is $400 million, yet funding for this effort is essentially nonexistent. Because Air Force leadership mandates Y2K compliance for mission-essential systems ahead of all other maintenance efforts, the effort is instead funded from sustainment funding or included as part of the current maintenance workload [2]. The problem is, maintenance and support programs require funding expansion to devise and execute new plans and tests for the Y2K transition. To set aside work and squeeze the extra effort in provides ever-diminishing returns while postponing other vital work.

The point is that it appears a large percentage of the $600 billion expenditure will be taken from participating organizations' current support funding and not from new funding initiatives. The full consequences from pulling resources from other programs to pay for Y2K conversion work may generate costly consequences yet unidentified. And no one can predict the cost of litigation, business failures, and other ripple effects from the eventual mass failure of interconnected computer systems. Although the exact return on investment is unknown for Y2K projects, it will almost certainly be far cheaper to fix problems than to ignore them.[1]

Simple Fixes Are Not Cure-Alls

There are many Y2K solutions floating among information technology professionals, many of which look feasible on paper but have limited practical application. One such solution is espoused by Peter Errington on pages 25-26 of this issue of *CROSSTALK*. It should be noted that his solution is not original and has been in use for some time, and it does work for some computer systems. Variations of his solution are evident in sliding window techniques planned or already used in many Y2K efforts we have reviewed.

Although Errington limited his solution to batch-processing computer systems (a severe limitation in today's automation data world), variations of his technique can and are being more broadly applied. However, his solution still retains the following limitations, which are similar to the limitations of other simple-looking solutions.

- Although windowing techniques (similar to Errington's solution) are often initially considered, the consensus is that they are adequate in only a small percentage of cases.
- As mentioned, the process of fixing program code or data accounts for only a limited portion of total Y2K costs. Roughly half of the Y2K effort occurs during the testing phase alone—and unfortunately, solutions like Errington's often ultimately require extensive testing. Other efforts include management concerns, planning, and integration, which also are outside of the scope of performing actual Y2K corrections. You cannot ignore, minimize, or partially complete the other phases of a Y2K solution without introducing detrimental risks.
- Job Control Language (JCL) code usually requires a significant amount of complexity or elaborate logic sequences to be useful for most systems. And even once implemented, such solutions often still do not adequately address many Y2K problems.
- The time required to develop adequate JCL solutions increases rapidly as the scheme to handle Y2K differences develops. This drives the need to design application-specific resolutions, especially as the level of the JCL complexity increases. The difficulty is recognizing the trade-off (thresholds) of when JCL solutions are impractical or insufficient.
- JCL is not robust enough to correct critical problems within databases and applications. This is usually because an application's logic flow cannot be externalized.
- Major components found in distributed systems or system interfaces require complex testing or other alternatives to gain the assurances required.
- Additional regression testing is always needed. These tests need to be enhanced to assure correct or adequate system performance.

The Testing Phase

Roughly half of Y2K costs are incurred during the testing phase. Insidious problems appear while testing many systems; often, these problems are discovered within applications that were thought to be fixed or already compliant. Another problem arises when a system or application that is determined to be nonessential later turns out to be essential, and no contingency plan exists for it.

As every programmer knows, even minor-looking program changes can cause unexpected outcomes that are time consuming to diagnose and fix. Approaches are being developed that minimize the scope of unexpected outcomes (see Don Estes' summary of encapsulation strategies on pages 9-10 of this issue); however, thorough testing is always needed to ensure that the program and data will perform as expected when the system clock has been

set past, and the data aged to, the year 2000.

For example, one organization used a system database that was developed with four-digit years, giving management unwarranted confidence that the system was Y2K compliant. When a test was conducted with the system date set to the year 2000, the application could not add any data to the database. This surprised management and developers, who were disappointed with the outcome but appreciated knowing the truth before the year 2000. Because testing was performed early, they had time to obtain an updated version from the vendor. Still, initially, all were confident that no Y2K problems existed. The old notion that "one good test is worth a thousand expert opinions" should come to mind under all Y2K circumstances.

The Y2K Cloud's Silver Lining

Systems with Y2K problems often need reengineering as well. The initiation of the Y2K effort may be a good time to obtain the information necessary for improvement. During Y2K efforts, objective data can usually be obtained to quantify a system's future. For example, the steps identified in the Software Reengineering Assessment Handbook [3] generally will clarify or make apparent the values for reengineering a system. However, prudence should be exercised to assure that extra activities do not dilute the Y2K testing or remedies.

The tools and improved methods spawned by the Y2K effort can be considered a healthy dose of medicine forced upon organizations. These methods are adopted because of the time constraints and the magnitude of effort that must be accomplished. Systems are undergoing inventory updates, system and file cleanup, documentation changes, and improvements to all processes to accommodate the Y2K issues.

Much of this work is corrective surgery, long neglected, or otherwise not possible. There is renewed focus on management and management techniques accommodating these efforts. New tools are used to assure a rapid and thorough test or renovation of the processes. The question then follows, will the use of these improved processes continue? If not, perhaps the organization's Y2K efforts will in the long run largely be a waste of money.

Conclusion

Easy-looking fixes to Y2K problems rarely offer significant cost savings. To shortcut system disciplines or depend on cure-all solutions is a negligent attitude that can potentially lead to disaster. To believe that a farce, racket, or ruse exists may cause you to be unrealistically satisfied that you are adequately treating your Y2K problems. Still, overdoing a Y2K effort is wasteful and may even be detrimental to a system's useful life. Organizations must strive to achieve and maintain a professional perspective and to take a balanced approach on all Y2K issues. ◆

Acknowledgments

I thank Paul Hewitt of the STSC and Gregory Daich of SAIC (assigned to the STSC) for their inputs to this article.

About the Author

Paul Harames is the lead of the STSC reengineering domain. He is an electronic engineer with over 31 years government

and industrial automation experience. His experience includes an extensive background in test and evaluation of radar equipment and radar target processes for air traffic control and airborne radar systems. He was an instructor at the Federal Aviation Administration Academy, teaching and developing courses for Air Route Terminal Automation systems. More recent work includes updates of automatic test equipment (microwave) and development of personal computer-based automatic test systems. He has a bachelor's degree in solid state physics from Oklahoma City University.

References

1. Cassell, J., K. Schick, B. Hall, and J. Phelps, "Time Marches On–Less Than 900 Working Days to January 1, 2000," Asia Pacific View (APV), June 28, 1996 (see http://www.gartner.com/forms/meyr2000.rsp.html).
2. Stevens, Capt. Chris, "The Air Force and Year 2000," *CROSSTALK*, STSC, Hill Air Force Base, Utah, January 1998, pp. 3-4.
3. "Software Reengineering Assessment Handbook," Version 3.0, JLC-HDBK-SRAH, March 1997 (see http://www.stsc.hill.af.mil under "Reengineering").

Note

1. Some have suggested that there has been significant savings (millions of dollars) from retaining two-digit dates. But what is the point? It is like saying that going out to eat is affordable because of a 50 percent-off sale at a clothing store where purchases were made earlier. It is irrelevant that millions of dollars have been saved unless this money had already been available to help resolve the Y2K problems.