Preparing for the New Millennium

Lt. Col. Joe Jarzombek
U.S. Air Force ESIP Director

A time bomb is ticking inside billions of lines of code: those two missing digits from the calendar year designation. Most computer systems designed in the past few decades refer to the year only by its last two digits (95, 96, 97, etc.). Originally a cost-cutting measure (because computer memory was expensive and programmers looked for any way to save space), this standard practice prevents systems from distinguishing one century from another. When the rest of us make the transition from Dec. 31, 1999 to Jan. 1, 2000, many unmodified systems will turn the clock back to Jan. 1, 1900.

We are now a scant two years away from that apocalyptic date. Despite warnings, technical descriptions of the problem, solutions, and widespread awareness, the question still remains: Are we prepared for that first millisecond after midnight on Dec. 31, 1999?

At a recent Air Force year 2000 (Y2K) conference, Lt. Gen. William J. Donahue stated that he has spoken to every Air Force four-star general and every functional chief regarding the Y2K problem and has concluded that

> "Some Air Force activities are in denial, others are on top of it. We will not fix everything, but we must maintain mission capability. The Air Force has $350 million for annual software sustainment. If you do nothing but fix Y2K with your funding, so be it."

William Ulrich and Ian Hayes, Y2K consultants, stated,

> "Procrastination, whether inadvertent or by design, is preventing many organizations from launching more than one or two projects at a time. By the time companies reach full-scale deployment, it may be too late to stabilize mission-critical systems. The reasons behind this delay include analysis paralysis, politics, the fear of mistakes, confusion, lack of budget appropriation, or just ignorance of the effects of the problem. ... The arrival of the year 2000 will be death by a thousand small cuts. A critical system failure may occur from time to time, but the more common situation will involve hundreds of inconveniences that pile up day after day." ("The Year 2000 Crisis: State of the Industry," *Software Magazine,* October 1997).

Many factors may contribute to a Y2K system failure, including denial, procrastination, fear, confusion, and budget constraints. It is interesting to note that many of these factors are not technical in nature, but instead are psychological or programmatic.

As we prepared this issue of *CrossTalk*, it was enlightening to see that some agencies are making real progress in becoming Y2K compliant, such as the Defense Logistics Agency (see p. 11). On the other hand, it was disheartening to hear from experts in the field that there are real risks in delivering Y2K projects on time (see "Throwing down the Y2K Gauntlet" by Peter de Jager, p. 5). The loss of a magazine's cover art is insignificant compared to the loss of accurate missile guidance systems.

The Department of Defense is using a five-phase approach to become Y2K compliant: Awareness, Assessment, Renovation, Validation, and Implementation. If you find yourself involved in one of these phases, perhaps this issue of *CrossTalk* will be an aid to you. You may also want to check the Web sites referenced on page 21. ◆

Letter to the Editor

Better Planning and Cooperation Needed for Open Systems to Work

I enjoyed the November publisher's note, "Open Systems Obstacles," by Reuel Alder, and I fully agree. The way our governmental agencies are organized leads one to falsely believe that open systems are the best solution. An area that greatly hinders the open system concepts, yet is used as the prime method to meet the future and foster the open system, is what I call "the power play" with and within organizations and management ("Ours must be first if we want approval.") Organizations don't want to lose their identity, and management does not want to share or lose their power base. Promotion, control, and budget often depend on this power base ("My open system is better than yours.") This encourages waste, nonquality products, and more open systems. It limits vision, planning, development, communications, and user acceptance.

So what can we do? For one, we can start to educate ourselves on the whole rather than our piece of the pie. We should use all resources available whether they belong to us or to someone else. (Yes, the commercial world can sometimes do it better, and two heads are better than one [usually].) We must establish measurable goals, a line of communication that encourages interfaces with vision and planning on the large, and be consistent in our vision and planning. We should stop being selfish and seek a known benefit(s) for every cost before we expend our resources, time, and budget.

Charles W. Locklin Jr.
Gunter Annex
Maxwell Air Force Base, Ala.

The Air Force and Year 2000

Capt. Chris Stephens
Air Force Year 2000 Program Management Office

The Air Force's Year 2000 Program Management Office is providing the necessary guidance to ensure all systems and infrastructure components that impact the Air Force mission be tested and made year 2000 compliant by the end of 1998.

If you are not familiar with the year 2000 (Y2K) problem, it is, in effect, the inability of many microprocessor-controlled systems to properly handle the transition from 1999 to 2000 ("99" to "00" in two-digit representation). The secretary of the Air Force and the chief of staff said in a June 24, 1997 letter that "the Air Force is arguably the most technologically dependent component of the United States armed forces" (*CROSSTALK*, December 1997). As such, this seemingly minor problem represents a potential threat to our ability to sustain our air and space mission. At this point in time, no one can determine absolutely the impact of this event on our mission capabilities. For the U.S. Air Force, unknown impact equals unacceptable risk.

Attacking the Y2K problem is a top priority for every Air Force organization. It must be understood that the problem is not limited to automated information systems (AIS) and weapons systems, but includes *everything* with a microprocessor in it: medical equipment, elevators, building entry-control systems, street lights, fire-suppression systems, and even fax machines (to mention but a fraction of the list). Individual failure of these items may be only a minor problem; widespread failure or degradation across the Air Force is something else entirely.

Tackling this problem is to be done using *existing* resources. That means resources—both dollars and people—must be reassigned to address year 2000. This is a monumental task given that the current cost estimate for the Air Force is more than $400 million. As we turn our attention to the cost of fixing our infrastructure, the cost will continue to climb. This realignment of resources requires the active participation of every leader in the Air Force if corrections are to be completed and in place by late 1998.

The Air Force established an Air Force Y2K Program Management Office at the Air Force Communications Agency (AFCA) at Scott Air Force Base, Ill. in February 1995. The goal of this program is to ensure that no mission-critical systems are adversely impacted by the year 2000. An infrastructure of dedicated professionals has been set up across the Air Force to tackle this problem. People at all levels are involved: major command (MAJCOM), field operating agencies (FOAs), direct reporting units, functionals, and base level. The program office has published several guidance packages to cover the different items that may be impacted and the roles and responsibilities of everyone. Everybody must be familiar with these documents and stay informed on the issues. If you do not know who your MAJCOM or unit point of contact is, give us a call. Everyone in the Air Force must understand this problem and be actively involved in minimizing its impact.

Philosophy

The enduring management philosophy for the year 2000 by the Office of the Assistant Secretary of Defense for Command, Control, Communications, and Intelligence has been centralized management with decentralized execution. The basis of this concept is that commanders command, system developers produce new capabilities, and lifecycle managers test, field, and maintain systems and software on a daily basis. Using this rationale, resident skills can be combined with processes to establish management structures, assign responsibilities, analyze Y2K defects, develop renovation strategies, replace or retire systems or components where necessary, and test and field compliant systems. The centralized management side of the equation is Air Force-level day-to-day management, policy, and direction necessary to ensure that Y2K changes are effectively and efficiently executed. The bottom line is that all systems and infrastructure components that impact the Air Force mission must be tested and made compliant in time.

Program Initiatives

The Air Force has developed processes and capabilities to efficiently manage the execution of the Y2K program. AFCA has issued direction in the form of three Air Force Y2K guidance packages:

- The Air Force Year 2000 Guidance Package is the Air Force plan for addressing AIS and weapons systems. This package details the five-phase process the Air Force uses to assess, analyze, and implement Y2K solutions.
- The Air Force Year 2000 Infrastructure MAJCOM/Commanders Guidance Package details a management approach to solve Y2K infrastructure problems.
- The Air Force Functional Year 2000 Guidance Package defines the Air Force functional communities role in resolving Y2K impacts.

AFCA has developed the Air Force Year 2000 Web site that is the official repository for all Air Force Y2K information at http://year2000.af.mil. This site contains links to other Office of the Secretary of Defense (OSD) and U.S.

government agency Y2K pages, e.g., the General Services Administration repository for commercial-off-the-shelf compliance information. AFCA has established the Air Force Year 2000 help desk, which is accessible via the Web site, by E-mail at afca-afy2k@scott.af.mil, or by telephone at DSN 576-5761.

AFCA developed the Air Force Automated Systems Inventory (AFASI) to inventory all Air Force AIS and weapons systems and to track their progress toward Y2K compliance. AFASI is the database used to track progress and update the Defense Integration Support Tools. Over 400 accounts have been established, and all MAJCOMs and FOAs are updating or validating Y2K information for systems for which they are responsible. The AFASI is used to prepare the status of the Air Force Y2K effort reports for Air Force Combat Intelligence Operations Center, Information Technology, who in turn prepares reports for OSD.

AFCA has developed a formal system testing and certification process and provides hands-on training to Air Force system certifiers who provide them with the necessary skills to test and certify weapons and AIS as Y2K compliant. The purpose of certification is to ensure that a system meets a minimum set of criteria to reasonably make sure it is not impacted by the year 2000; the implementation of this program ensures that certification is applied consistently and accurately across the entire Air Force.

AFCA conducts periodic Air Force Y2K working groups that draw together command Y2K representatives to discuss issues and resolve problems. The sixth working group meeting was held Aug. 5-6, 1997. It was the first session with colonel representation from the commands. This session truly raised the awareness levels in the commands.

How to Contact Us

If you have any questions or concerns or would like to know more about the Air Force's Y2K program, contact the Air Force Year 2000 Program Management Office (AFCA/ITY) at 618-256-5697 DSN 576-5697 or visit our Web page at http://year2000.af.mil (government access only). There you will find the Air Force Y2K guidance packages, information on our testing and certification program, and reference to many other Web sites throughout the Air Force, Department of Defense, government, and industry. ◆

552nd CSG Receives CMM Level 3 Rating

Cheryl Stefenel and Gordon Fitzgerald
552nd Computer Systems Group

After 18 months of hard work and dedication, the 552nd Computer Systems Group (CSG)—communications and computer support for the 552nd Air Control Wing—recently achieved Level 3 according to the Software Engineering Institute Capability Maturity Model (CMM) criteria.

In 1993, the 552nd CSG was assessed at Level 1, the starting point for all organizations. According to the group commander, Col. Frank Richardson, when the group was assessed at Level 1, many people thought the CSG would never achieve a Level 2, much less a Level 3.

Lt. Col. Vincent Azzarelli, former 552nd Computer Systems squadron commander who led the CMM Level 3 efforts agreed. "After the satisfactory rating the group received in the 1996 Quality Air Force Assessment, there wasn't a lot of confidence—the thinking within the group was that this would be too much of an uphill battle to accomplish in a year," he said.

However, once the group got motivated, they not only reached Level 2—their original goal—the Air Force Communications Agency assessed them at Level 3, which surprised the group.

"I am extremely proud of what the 552nd Computer Systems Squadron did to lead the CSG and the wing to a Level 3 rating. To move from Level 1 to Level 3 was just an awesome feat," Richardson said. "In addition, the support provided by the Air Combat Command Computer Systems Squadron and the Oklahoma City Air Logistics Center was invaluable."

After achieving the Level 3 rating, the CSG will improve its Level 3 processes, stabilize these processes, and get its Level 4 rating. The group hopes to have a Level 4 rating within the next two years.

Cheryl Stefenel
552 CSS/SCWA
Tinker AFB, OK 73135
Voice: 405-739-7524 DSN 339-7524
Fax: 405-734-4372

Throwing Down the Year 2000 Gauntlet

Peter de Jager
de Jager & Company, Ltd.

The year 2000 project is a large software project with a deadline that cannot be missed. Have we, as an industry, made a point of communicating the real risks that surround the delivery of the year 2000 project on time?

The year 2000 (Y2K) problem has many unique aspects, not the least of which is its size. So I think it is fair and useful to trim all those other aspects away and consider it as nothing but a *large project*. In other words, this is not a treatise on any of the technical issues associated with the year 2000. It is simply an observation about the information technology (IT) industry and how we have handled large projects in the past.

Many computer experts are loathe to dwell on the following observation about large projects, and certainly, we do not appear to be going out of our way to communicate it to the people (management) who must understand it.

"The computer industry has proven itself unreliable in the past when it comes to delivering projects on time."

This is a kind and gracious way to state that when it comes to delivering projects on time, the IT industry must be considered an abysmal failure. To drive this point home, the following is a recap of an audience encounter I have repeated dozens of times worldwide over the past year.

Request to audience of IT professionals: "Raise your hand if you have a high degree of confidence in your ability to deliver the year 2000 project on time."

Response from audience: A forest of hands raised in affirmation.

So far so good. This reflects what they are communicating to management: "Don't worry, we have this under control. We can handle this; we'll deliver it on time—trust us."

Next request: "Raise your hand if, over the past three years, you have delivered 100 percent of your applications on time."

Response: A gale of laughter. Why? Because the notion of 100 percent on-time delivery is as foreign to the IT industry as the notion that airlines can deliver 100 percent on-time departures.

Next request: "Raise your hand if your historical record of on-time delivery is 90 percent."

Response: At best, one hand will be raised, only to be lowered quickly in submission to roaring laughter and catcalls of "Liar!" (The hand is lowered even more quickly if these catcalls are coming from users in the audience.)

Next request: "Raise your hand if your historical record of on-time delivery is 80 percent."

Response: Two percent to 3 percent of audience might raise their hands.

Next request: "Raise your hand if your historical record of on-time delivery is 70 percent."

Response: Another 2 percent to 5 percent of audience might raise their hands.

It is not until I get to between 50 percent and 60 percent on-time delivery that half the audience members have raised their hands, which means the historical track record for half the audience is below 50 percent on-time delivery. Instead of asking for a show of hands for each of these requests, I could instead just provide an industry figure: 86 percent of all applications are delivered either late or never.

But remember that when I asked if these same people had a high degree of confidence in their ability to deliver this Y2K project on time, they gleefully gestured "Yes!" This raises an ironic question of intense interest to management: Why are we so confident about our ability to succeed in the future when we have failed so miserably in the past?

A Gamble

I then ask the audience if there are any gamblers in the room. I ask them if they would like to play a little gambling game with me. I explain that a gamble has three components: the ante, the event, and the payoff. I ask them if they would put $1,000 of their money in my left hand (the ante). I will flip a coin (the event). If it is heads, I will give them back their $1,000; otherwise, I will keep their $1,000 (the payoff). The response is naturally another gale of laughter. It is a sucker's bet.

I then point out we are all already playing the game, except the stakes are higher—much higher. The ante is your organization. The event is your proven ability, not your wishful thinking, to deliver large projects on time. The payoff is the ability of your organization to function in the year 2000.

The Gauntlet Is Dropped

So here is the gauntlet being dropped at our feet: Have we, as an industry, made a point of communicating the real risks that surround our delivery of the Y2K project on time? Or are we trying to placate management? "Don't worry; be happy. Everything is all right. No need for alarm"?

We have done it before—many times. The following report from the Sept. 16, 1997 *New York Times* concerns the cancellation of a $100 million IT project that should have been delivered in September 1997.

> "Until a few months ago, Medicare officials were consistently upbeat in their public statements about the new computer system and brushed aside the skepticism expressed by the Congressional Auditors."

One could point out this was not a Y2K project, but does that matter? They are all big projects, and surely they obey the laws of large projects. Surely, they are all affected by Murphy's Law to the same degree.

To be sure, the Y2K project *is* different—it has a deadline we cannot miss. But I do not know if a "real" deadline, one that cannot be adjusted regardless of the size of the task, will increase or decrease the likelihood of success. I suspect it will decrease our chances; we will see.

Are you being consistently upbeat to a fault? Are you guilty of communicating that all is well when you know otherwise? If you had to answer the requests posed in this article before an audience, I doubt you could not worry about your past success rate. So pick up a pen—IT workers should write down their organization's proven historical track record for delivering projects on time. Then, you can face up to the following questions:

- Have you communicated your organization's success rate to management?
- Do they understand the risk the organization is undertaking?
- Do they understand the consequences of failure? Do you? ◆

The Tenth Annual

SOFTWARE TECHNOLOGY CONFERENCE

ÒKnowledge SharingÑGlobal Information NetworksÓ

19-23 April 1998

Co-sponsored by:

United States Air Force	Defense Information Systems Agency (DISA)
United States Army	Utah State University Extension
United States Navy	
United States Marine Corps	

Salt Palace Convention Center
Salt Lake City, Utah

For further Program and Exhibit information or to register

Conference Management—voice 801-777-7411, DSN 777-7411, dovenbar@oodis01.hill.af.mil

Conference Administration—voice 801-797-0787, dallas@stc.usu.edu

STC '98 Website—http://www.stc 98.org

Conference Presentation Topics Include:

- Capability Maturity Models
- Client/Server and Distributed Computing
- Configuration Management
- Cost Estimation
- Data Administration
- DII
- DoD Software Policies
- Education and Training
- Embedded Software
- Global Information Issues
- Inspections
- Internet/Intranet
- Knowledge Based Systems
- Measures/Metrics
- Object-Oriented Technology and Languages
- Open Systems and Architecture
- Outsourcing & Privatization
- Process Improvement
- Product Line Engineering
- Project Management
- Quality Assurance
- Risk Management
- Security
- Software Acquisition
- Software Engineering
- Software Maturity
- Technology Adoption
- Y2K

STC℠

About the Author

Peter de Jager has been active in bringing the Y2K problem to the awareness of both the information systems community and the business world at large. He is perceived by many to be the world leader in creating awareness for the Y2K computer crisis. He has written numerous articles on the subject, including the ground-breaking "Doomsday 2000" article published in *Computerworld*, Sept. 6, 1993. He has presented the Y2K problem to technical and general audiences in Canada, United States, England, Holland, Finland, Norway, Sweden, and South Africa. A Canadian citizen, he was summoned before the U.S. House of Representatives Science Subcommittee hearings to testify on the Y2K crisis. He acts as a special adviser to the United Kingdom Year 2000 task force and was recently appointed as special adviser to the Russian task force. He also is a contributing editor to *CIO Canada* and a management columnist for *Information Canada*. In January 1997, he began writing the monthly Y2K column in *Datamation* and is a contributing editor to that magazine. He created the Year 2000 Information Center, which has more than 180,000 home page accesses on the site each month.

de Jager & Company Limited
Voice: 905-792-8706
Fax: 905-792-9818
E-mail: pdejager@year2000.com.
Internet: http://www.year2000.com
Speaking schedule: http://www.hookup.net/~pdejager

Software Configuration Management Helps Solve Year 2000 Change Integration Obstacles

Tom Burton
InRoads Technology

While defense organizations implement year 2000 (Y2K) changes, they continue to implement many day-to-day changes on these same applications. These two activities are usually undertaken separately, resulting in the potential for disjointed development efforts and software crises. Successful organizations will need to manage Y2K conversion and other changes through effective software configuration management.

The defense software engineering community is grappling with the challenge of changing code in mission-critical applications so that they will work properly in the next century. This conversion saga requires the implementation of thousands of changes to application areas that have not been touched for years.

It is laborious and time consuming to find areas where code must be changed. Engineers, for instance, must wrestle with missing source code, wrong versions, redundant modules, and sloppy documentation. They also must deal with the thousands of additional fixes that have to be implemented throughout the conversion project. It is a constant worry that unwanted changes will fall unnoticed between the cracks.

Yet, while date-change activities are taking place, software engineers must conduct day-to-day revisions to the same applications. Such modifications may be implemented to remove bugs and add new features, functionality, or enhancements to these applications.

The scope of these day-to-day changes also is substantial; the code of mission-critical applications is constantly being modified for various reasons. For instance, different versions of defense applications often must be built to accommodate a multitude of simulation scenarios, e.g., it is much cheaper to test a missile's software under various simulation scenarios than to conduct actual launches for the same purpose.

Software engineers must constantly modify an application as it goes through various lifecycle stages. Additional challenges include updating documentation while the changes are in progress, making changes to firmware, and factoring hardware changes into the equation. None of these revisions are trivial given the complexity of today's mission-critical applications. If not properly managed, these changes may cause the system to collapse.

Often, the problem is that the conversion process and the day-to-day application changes are treated as separate projects, and therefore implemented by multiple teams without effective coordination. The development environments in such organizations could face a crisis of gigantic proportion with the turn of the new century.

The Y2K conversion project and day-to-day changes must be conducted harmoniously, since work may involve thousands of changes to millions of lines of code. Otherwise, changes may be accidentally left out of the end-user's version—an unacceptable mistake. Lives can be lost if, for instance, an automated combat response system triggers an attack against a friendly ship or aircraft. Developers cannot afford to deliver software that is only *mostly* the right version.

The Software Configuration Management Difference

Fortunately, development organizations can potentially deliver 100-percent-right software by placing Y2K conversion and day-to-day development activities under the common umbrella of software configuration management (SCM), also commonly referred to as process configuration management. SCM strictly organizes the tasks and activities that maintain the integrity of software product configurations, ensuring configurations are correct, i.e., that engineers are working on the right source code and the right versions of an application.

Many development organizations, whether they know it, spend 25 percent or more of their time trying to manage their configurations. The good news is that the defense industry is ahead of the pack for having recognized the need for automated SCM long before it became an established concept in the commercial sector. The bad news is that—although the importance of SCM has been recognized for years and millions of dollars have been spent on SCM tools—few software development organizations do a great job of SCM.

In fact, most average software development projects are only able to keep 75 percent to 80 percent correct software configurations. Many are lucky to maintain a 50 percent to 60 percent correctness, and quite a few projects have software product configurations less than 50 percent correct [1]. We should not expect better results for Y2K conversion projects unless we implement better controls than the current state of the practice. Considering the critical need for Y2K conversions to be precise, there is an obvious need for most organizations to become much better at SCM.

Good SCM

Good SCM occurs when the configurations are continually 100 percent correct—there are no lost or missing changes, all the correct software components and versions are included in the builds, and no changes targeted for the next release somehow end up in the current release. Good SCM also provides proof that the configurations are 100 percent correct at any time in the development lifecycle.

In sum, good SCM is achieved when the software product configurations are 100 percent correct and contain all the wanted changes, when the development organization is 100 percent certain that these applications are complete, and when the development team can demonstrate that these applications are 100 percent correct.

But how does an organization get to that point? To be effective, the Y2K conversion project and the day-to-day development activities must be implemented concurrently and in parallel under the umbrella of good SCM. This means that software engineers must take all the changes from the Y2K projects and all the changes from the day-to-day projects and integrate them together, test them, merge them, and not leave anything behind.

SCM Pitfalls to Avoid

It is difficult to achieve good SCM without glitches. Three main issues significantly impact the ability of the organization to establish SCM.

First, poor SCM often results in changes being made, but then not put back into the product configuration. This happens when programmers forget that they have made a particular change, then the developer reinserts the wrong version of the program. This typically occurs when the programmer does not understand or follow the organization's established SCM processes.

A second problem can occur when parallel and concurrent activities are taking place: one developer's changes to a version overwrite another developer's revisions to the same version. This is known as "change regression," and though common, this problem is often overlooked. Such a scenario typically occurs when there is no automated process CM tool in place that can track all changes to the system, allowing developers to compare their changes and select the ones they want to keep.

A third SCM problem occurs during the build process, when source programs are compiled and linked to create executable programs. Because the build process is typically automated, wrong versions can easily be incorporated automatically into the executable program. This happens when developers do not know from where the build is picking the executable program; therefore, they create and test incorrect executables. Many additional SCM issues also come into play during the build process, all of which impact the integrity of the software product configurations.

Implementing Good SCM Practices

Three critical practices can prevent these pitfalls and promote good SCM, helping achieve the goal of 100 percent correct mission-critical applications to users.

- The organization must formally document its development processes and use these processes as a road map to effectively integrate the Y2K projects and the day-to-day changes to these applications.
- The team must be educated as to what constitutes good SCM practices. Training should be viewed as a process that spans the development lifecycle. For instance, training can teach the team the organization's processes, help shorten the evaluation of SCM tools, help deploy SCM tools, and facilitate implementation. Many outside resources can help achieve SCM competency. Some vendors even offer ready-made SCM classroom materials and multimedia computer-based courses, which can effectively be used for in-house training.
- The development organization would do well to rely on leading-edge process configuration management tools, which help automate, distribute, and merge the changes associated with the Y2K project and day-to-day development activities. These client-server solutions provide facilities to map organizational and development processes into the tool. They also provide a complete audit trail of all development activities so that the development environment can guarantee that the applications delivered to customers are 100 percent complete.

Conclusion

Organizations that treat Y2K conversion projects separately from day-to-day development activities are likely to experience significant trouble and setbacks. To start on a path toward 100-percent-correct software, these organizations will have to put these two efforts under the common umbrella of SCM. Achieving good SCM is possible in this framework when software processes guide the overall development effort, when developers are educated about effective SCM practices, and when these efforts are automated via a leading-edge process CM tool. This will put the software development environment on a more secure path to success. ◆

About the Author

Tom Burton is CEO of InRoads Technology, which specializes in providing tools and education for the successful implementation of software configuration management. He has more than 10 years experience in software configuration management.

Voice: 805-967-4545
Fax: 805-964-4790
E-mail: tburton@inroadstech.com
Internet: http://www.inroadstech.com

References

1. "Configuration Management Industry Outlook," *InRoads Technology*, January 1997.
2. "The Year 2000 Digit Crisis Sounds the Alarm for Active Control of Software via Process Configuration Management," white paper by Tani Haque, CEO of SQL Software, November 1996, info@sql.com.
3. "Creating a Culture for Successful Process Configuration Management," white paper by Tani Haque, CEO of SQL Software, July 1997, info@sql.com.

Encapsulation Solutions for Year 2000 Compliance: A Summary

Don Estes
2000 Technologies Corporation

This article discusses two variations of time-shifting strategies for year 2000 compliance: data encapsulation and program encapsulation. A summary of the complete article follows; the detailed article, containing specific strategies and examples, can be found on the CROSSTALK *Web site at* http://www.stsc.hill.af.mil/CrossTalk/crostalk.html.

Standard technical strategies to achieve year 2000 (Y2K) compliance include replacement, date expansion, and various forms of windowing. Recently added to this list are time-shifting strategies, which significantly reduce costs, business risks, and time to implement and test. All of the benefits of encapsulation result from two central facts:

- Analysis and implementation efforts are minimal.
- It is the inverse of the procedure used to age data into the future to establish future data compliance; once one successfully proves current-year regression testing, there is an implicit establishment of future-dated regression testing for the length of time constant employed.

A different implementation of the encapsulation concept can be employed as a test harness to perform automated Y2K testing, including dynamic future date data aging. A description of the automated testing implementation is planned for a future CROSSTALK article.

Time-shifting strategies dramatically reduce the costs of testing by eliminating the need to build future-dated test cases, and because they use a complementary automated regression testing facility, they are bound into the program logic with the encapsulation logic. The risk profile also is minimal because the relatively small amount of affected program code is at the boundary between the program and the outside data storage. Because the file formats do not change, it also has minimal implementation and deployment impact.

The automation of date expansion and procedural logic solutions offer many of the implementation advantages of encapsulation, including cost reduction. Limited windowing methods can compete with encapsulation on an implementation cost basis for many applications. However, in the area of testing, encapsulation is in a class by itself. Only encapsulation can bypass the expense of future-dated testing altogether and, through automation, bypass the manual construction of unit test data. As the event horizon draws near, this aspect of encapsulation will dominate strategy decisions.

Data Encapsulation vs. Program Encapsulation

The difference between data and program encapsulation is in what you change. The mnemonic rule is you encapsulate what you do not change. So, if you are changing programs but not data, you are performing data encapsulation. But if you are changing data but not programs, you are performing program encapsulation. A hybrid of the two interposes a layer between program and data to perform the time shifting.

Encapsulation strategies are similar to windowing strategies in that a two-digit year is maintained. However, procedural logic strategies infer the century from the data and operate while spanning the century boundary. Time-shifting strategies, by contrast, shift the data back in time to avoid the century boundary altogether. The essential problem with maintaining a two-digit year is that 2000 > 1999 but 00 < 99. By shifting the dates back in time, typically by a multiple of 28 years, we end up with 1972 > 1971 and 72 > 71, which solves the problem. As long as all dates are shifted consistently, we receive the same results for the same input, and the applications will work until 2027 or 2055. Once all stored data are in the 21st century, the time shift can be turned off, at which time the application will work until 2100. This can be considered a permanent fix.

One absolute requirement with this method is that no two-digit years can be stored from before 1929 (for a 28-year shift) for any date used in a comparison or a calculation, although there are special case exceptions to this rule. The reason is that once shifted, the date data must all lie in the same century. This requirement does not apply to dates used merely for storage and retrieval.

Encapsulation is unique because the indeterminacy of assessment is eliminated; this is the major source of delay, even in windowing and expansion projects using automated assessment tools of great power. The single data entry and exit points in each program are vastly fewer in number and are essentially decoupled from each other, so that one need only examine those points and the data flowing through them to answer all questions required for encapsulation. If any doubts remain, one need only dump the relevant data files or tables and look at them.

Data encapsulation works on a program-by-program basis, as compared to program encapsulation, which works on a system-by-system basis. As a result, within the same system it is possible to use data encapsulation for one program, a standard windowing solution for another, and to leave a third completely unchanged because date data flows through the program without being processed in any way. This may be important for sites that plan to use encapsulation as a short-term fix while preparing a more comprehensive solution via windowing, expansion, or replacement.

Program encapsulation may have an implementation advantage over data encapsulation, although this will be significant primarily in larger projects. This is because programs that do not cross the time-warp zone boundary usually do not require modification.

Encapsulation Metrics

Data encapsulation was first proposed by a major defense contractor in 1992, and we are now aware of some two dozen pilot and full projects that use the method. In addition, both program and data encapsulation can now be automated. Early metrics show an average of 1,000 to 2,000 lines of code per day per programmer for manual data encapsulation implementation, and 10 times this for automated implementation. Program encapsulation can be even more efficient, particularly for larger projects.

Conclusion

Encapsulation, although a new strategy relative to expansion, replacement, or windowing, is rapidly proving itself as the most efficient in terms of time and cost and will increasingly be the center of consideration as we move toward our time horizon for failure. ◆

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 auto-

mated testing system. He also works closely with vendors of limited windowing, 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, 1-800-756-8046
E-mail: info@2000technologies.com

Call for Articles

If your experience or research has produced information that could be useful to others, CROSSTALK will get the word out. Not only is CROSSTALK a forum for high-profile leaders, it is an effective medium for useful information from all levels within the Department of Defense (DoD), industry, and academia.

Published monthly, CROSSTALK is an official DoD periodical distributed to over 19,000 readers, plus uncounted others who are exposed to the journal in offices, libraries, the Internet, and other venues. CROSSTALK articles are also regularly reprinted in other publications.

We welcome articles on all software-related topics, but are especially interested in several high-interest areas. Drawing from reader survey data, we will highlight your most requested article topics as themes for 1998 CROSSTALK issues. In future issues, we will place a special, yet nonexclusive, focus on

Internet/Intranet
June 1998
Article Submission Deadline: Feb. 2, 1998

Project Management
July 1998
Article Submission Deadline: March 2, 1998

Measure and Metrics
August 1998
Article Submission Deadline: April 6, 1998

Look for additional announcements that reveal more of our future issues' themes. We will accept article submissions on all software-related topics at any time; our issues will not focus exclusively on the featured theme.

Please follow the *Guidelines for CROSSTALK Authors*, available on the Internet at http://www.stsc.hill.af.mil/. Hard copies of the guidelines are also available upon request. All articles must be approved by the CROSSTALK Editorial Board prior to publication. We do not pay for articles. Send articles to

Ogden ALC/TISE
ATTN: Heather Winward, Crosstalk Features Coordinator
7278 Fourth Street
Hill AFB, UT 84056-5205

Or E-mail articles to winwardh@software.hill.af.mil/. For additional information, call 801-777-9239.

Tracy Stauder
Managing Editor

Defense Logistics Agency's Year 2000 Program
Managing Organization-Wide Conversion and Compliance

Sarah J. Reed
Defense Logistics Agency System Design Center

The Defense Logistics Agency considers the year 2000 (Y2K) problem mission-critical, and we have treated it as such in planning and executing the largest maintenance effort we have undertaken. The agency kicked off a formal Y2K project in November 1995 with nearly a full year of planning, preparation, and piloting. This article discusses our Y2K initiative and our experiences in raising awareness and in assessing, renovating, and validating our systems. Our program, built on available industry research and our experiences, has been modified as we have gained fresh insight and assimilated new lessons learned.

The Defense Logistics Agency (DLA) maintains 40 million lines of code within 125 automated information systems. Over 80 percent of our major systems make use of date data in some way; half of those date references are high impact; that is, likely to adversely affect comparisons, calculations, or sort processes. DLA's System Design Center (DSDC) is a fee-for-service activity, and funded as we are by disparate functional proponents, we have had rare occasion to conduct a maintenance effort of this magnitude *across* multiple system structures. The degree of planning and coordination called for to accomplish Y2K-compliant systems has been for us, unprecedented. We began our formal Y2K project in November 1995.

System Impact Assessment

The purpose of conducting an organization-wide Y2K impact assessment was to ascertain the magnitude of the Y2K problem and to define and prioritize the remediation requirement. The assessment began in January 1996 with the development of a questionnaire from a Y2K project kick-off session and from industry and government resources. After several revisions, we had 27 questions about date practices, levels of customer awareness, staff support and the application environment (available in the Internet version of this article and on a secured server at http://www.dsdc.dla.mil/priv/projects/year2k/year2k.html). We then identified points of contact for system-level interview and survey. Responses were entered into a database, scrubbed for consistency, then queried and analyzed. Data analysis began in mid-April 1996 and ended with publication of the final report in late June 1996. The extensive data collection and analysis effort resulted in a fairly comprehensive application portfolio profile for DLA standard automated information systems (AIS).

Our Application Profile

A summarized look at some of the component data provides an idea of our size:

Component	Total number
Lines of code	39,577,427
Programs	60,060
Screens	33,416
Reports	8,905
Files	236,271
Database tables	10,379

We have 77 languages, the top languages being COBOL, C, and Assembler. Top database systems are Oracle, Rdb, and Unify. We use 35 different hardware platforms, 16 operating systems, and 311 commercial software packages for application development and maintenance.

Findings, Recommendations, and Resulting Actions

The analyzed systems fell into the following risk classifications determined according to the various remediation actions appropriate for them.

Expected to remain in place beyond 2000, but not compliant (47.1 percent, 56 applications). We addressed this critical risk category by developing funding proposals, conversion strategies and plans, and processes and tools to facilitate analysis, renovation, and testing. Conversion progress is assessed on a regular basis through status reports, progressed project plans, and metric analysis. These actions are intended to mitigate the risk of project slippage on Y2K conversions.

Expected to be rehosted, reengineered, or replaced but not compliant (16 percent, 19 applications). If replacement initiatives experience schedule slippage, Y2K failures could occur within the existing systems targeted for replacement. We regularly track replacement schedules so that we can recommend the initiation of renovation or contingency planning should the replacement system initiatives experience schedule slippage. Contingency planning, an omission perhaps in the early phases of our project, has increasingly become a concern. We now acknowledge the need to plan system-level approaches to fix Y2K problems in a system before its scheduled conversion and deployment has occurred and business area proponents began development of these plans in June 1997. The replacement system progress must also be monitored to ensure Y2K compliance before deployment.

Expected to remain in place beyond 2000 and compliant (33.6 percent, 40 applications). Systems thought to be compliant were perceived to be at less risk than those that were not compliant. However, we insist that system compliance statements be supported with the submission of a certification checklist that describes the system and the Y2K-related testing effort.

Related Observations and Recommendations

AISs not included in the survey data. Several systems were dropped from the survey for a variety of reasons, e.g., they were no longer in existence, are now maintained elsewhere or were too early in development to be profiled. These systems were rechecked a year later to ensure no omissions had occurred.

Database Management Systems (DBMS). The 15 different database systems in use have date types and uses that may or may not be Y2K compliant, depending on how they are used. We discovered that having a date field with a four-position year standard in a DBMS does not make an application that is integrated with it automatically compliant. We have investigated and documented our findings relevant to the date-handling procedures for various DBMSs. Where the results of our assessment reveal noncompliant commercial DBMS products we plan to discontinue its use or obtain a version upgrade, as appropriate.

Interface recommendations. Initially, the Y2K Program office maintained that all interfaces should be identified, described and documented, then negotiated to determine format transition actions. A vociferous outcry from the project managers of the largest AISs and advice from our executive Configuration Control Board (CCB) convinced us to concentrate resources on renovation of the system and filter building to protect our systems from noncompliant incoming data—and not on self-initiated interface format changes. Ultimately, we did agree that an interface strategy was necessary even if our base premise would be not to change the interface formats until receipt of a specific requirement. This strategy was delivered in August 1997.

Release Management. AIS upgrades are normally released to multiple deployment sites. Y2K considerations presented the additional challenge of simultaneously implementing Y2K releases, nonyear-2000 releases, and in some cases, mixed releases. To facilitate adequate implementation planning, we worked to establish a liaison with our chief deployment site.

Date formats. The wide variety of date formats (48) used among the AIS points out the need for greater standardization of date formats, but conversion to four-position year formats is not always practical for large, complex legacy systems given the risk and time constraints associated with them. Because of the variety of date formats and the prevalence of two-digit years, we recommended common date modules for processing and provided these modules for our more prevalent languages.

Vendor product compliance. The application impact assessment highlighted the need to further investigate the compliance of hardware platforms and operating systems. For our organization, compliance status of these commercial products must be established with the product vendor. The Y2K Program office continues to pursue the issue of vendor product compliance and to publish its findings along with appropriate recommendations.

Awareness. In spite of briefs to senior officials, we felt that perhaps customer and user communities were still largely uninformed even as they began funding the initiative in January 1997. We subsequently began to develop white papers and briefings that discussed aspects of interest to the user community. In addition, we now have regular project forums to facilitate the sharing of progress, findings, and concerns throughout the development organization.

Assessment process. We knew going into the assessment process that developing good survey questions, properly targeting the survey, and effectively analyzing the results would be a difficult task for our team, none of whom had a statistical analysis or survey science background. We were also concerned about the quality of the survey responses, often subjective and estimated. Though satisfied with our results and subsequent recommendations that guided our later efforts, we would recommend that inclusion of a team member with a statistical science background would be of benefit during impact assessment.

Vendor Product Risk Assessment and Mitigation

In April 1996, we laid out the following high-level plan for conducting a vendor product assessment:

- Send letters to all vendors for whom we had products under maintenance and ask for a statement regarding Y2K compliance status of the products we held.
- Enter response data into a repository.
- Analyze for risk classification.
- Recommend appropriate mitigation actions.
- Track and report outcomes and status.

Although almost 90 percent of the vendors responded, we began to discover a need to collect much more information. We had collected some vendor product information during the impact assessment; however, not all respondents provided clear or complete information about the commercial products integrated with their applications. This prompted a call for additional information.

Identification of Development Environment Software

In November 1996, we provided our first report on Y2K compliance in our vendor products. The resulting recommendations were several, but the chief recommendation was to expand upon our original data collection effort. We still could not confidently describe the risk in our hardware and system software, because we did not have an adequate baseline picture. Where we had expected to find a similar identification of products between our organizations, we found that of 74 products identified on our chief mainframe machine, 53 were *not* previously identified by the application support areas or the acquisition office. We established possible reasons for the discrepancies, then arduously worked to resolve them.

Where Are We Now?

There were 276 products identified as lacking complete compliance information. If the lack of information was because we could not properly identify the vendor, the AIS support group was contacted to help identify the product vendor to obtain the required information. Vendor contact was established or re-established to obtain new or additional information product information.

Improve software portfolio management processes. The great discrepancies between the view of what is owned, installed, and used was of such concern to us that we launched a major software portfolio management improvement effort, wider in scope and more permanent in its legacy than the Y2K effort.

Follow up with vendors expected to provide compliant products at a future date. For these products, we notify groups who support applications integrated with these products, because the projected date of compliance may be unacceptably late. We also make them aware of product upgrade impacts upon their application. Upgrade strategies will be developed on a case-by-case basis to manage application impact and ensure timely arrival, installation, and testing of the upgraded product.

Develop independent strategy for IBM products. A planned operating system upgrade should replace several, but not all, currently held noncompliant products. It was determined that the Year 2000 Program and OS/390 upgrade project team would need to work closely together to accomplish the OS upgrade and Y2K integration testing in a timely manner.

Develop testing strategies for those products vendors have stated are compliant. Ascertaining the state of vendor products as we were from direct vendor replies to our questions, research of vendor-supplied World Wide Web statements regarding Y2K compliance, and in some cases, relying on third-party-published studies still leaves a vulnerability regarding the actual performance of the vendor product. We have deter-

Table 1. *Y2K support tool categories according to DLA internal priorities.*

Priority	Tool Capability	Tool Category	Prioritization Rationale
1	Data Generation	Validation	Preparation of test data for year 2000; expected to be a significant effort via the automated generation of test data
	Test Case Management	Validation	Test case management can reduce effort to develop current century, cross-century, and next century test cases.
	Record/Playback Capability	Validation	Recording inputs for future playback reduces re-testing efforts.
	Data Aging	Validation	Preparation of test data for Y2K testing, via the ability to advance dates in test data transactions.
	Clock Simulators	Validation	Ability to simulate dates other than current system date.
2	Bridging (file conversion)	Renovation	Use of tools to reduce the bridging effort.
3	Test Coverage Analysis	Validation	Quantifies program logic execution during test.
4	Field Expansion	Renovation	Lower priority; our strategy utilizes expansion on limited basis due to associated cost and risk.
5	Code Generator	Renovation	Lower priority because capability already exists .
6	Version Control	Config. Mgt.	Capability available.
	Change Control	Config. Mgt.	Capability available.
7	Impact Analyzers	Assessment	Capability available.
	Cost of Work Estimating	Assessment	Capability available.
	Date Subroutines	Renovation	Capability available.
	Automated Conversions	Renovation	Capability available.
8	Version Merging	Renovation	Capability available.
	Change Tracking	Config. Mgt.	Capability available.
	Date Finders	Assessment	Capability available.
	Cross-References	Assessment	Capability available.
	File Comparisons	Validation	Capability available.
9	Data Name Rationalization	Assessment	Capability available.

mined that at least minimal spot testing of in-house versions of products stated to be compliant is prudent.

Review and Selection of Support Tools

The objective of this task was to recommend a tool set for use in performing Y2K assessment, renovation, and validation. Several sources were used to identify potential tools and determine appropriate tools for further investigation —product literature, Web sites, expos and conferences, industry-developed tool reports, and demonstrations. The following requirements were identified:

- Where possible, the tools should be from a major vendor to address concerns about tool maturity and vendor support.
- The tools should primarily address the major languages and platforms used within the DSDC. These were determined to be primarily COBOL running on the MVS platform, with C a distant second.
- Use of the tools should not require an entirely new method; they should require minimal training for effective use, and execute in the environment to which the development staff is accustomed.

Support tool categories are described in Table 1. The higher priority tools (1, 2, 3, etc.) are those tools for which little or no existing internal capability existed at the time of the assessment.

Tool Recommendations and Implementations

Most identified tool needs could be met with tools already owned. Release management changes were recommended to ensure all development sites had the latest tool versions. New tool recommendations were prioritized, and tools were brought in for environmental testing and evaluation prior to purchase. A Y2K analysis and remediation workshop was developed primarily for COBOL programmers, with emphasis on the procedural logic or date windowing solution.

Lessons Learned

- Outdated versions of system software in the development environment limited our ability to fully evaluate Y2K tool offerings.
- Lack of a clear description of the current or targeted development environment made planning for the incorporation of targeted Y2K tools into a long-term engineering strategy difficult.
- Once a product was obtained, it was sometimes difficult to get it installed. We were unable to complete the testing of software during the contract period of performance, and it was frustrating to know how helpful these tools could be to the development staff in accomplishing the Y2K work.

Certification Process to Benchmark and Report System Compliance

Organizational confidence in Y2K compliance statements requires a common certification process and benchmark. As with the impact assessment, a process was initially brainstormed at our first team meeting, then evolved. The draft guidance and certification checklist was piloted on systems believed to be compliant. The pilot results and emerging Department of Defence (DoD)-level guidance caused us to revise the process and checklist before submission to the executive CCB (available in the Internet version of this article and on a secured server at http://www.dsdc.dla.mil/priv/

Table 2. *Characteristics of six DLA Y2K conversion projects.*

Name	Platform Characteristics	Tools Used	Strategy Used	Pilot Size (Lines of Code)	Cost per Line of Code
EMACS	Distributed; 486 UNIX PCs. Unify DBMS, ACCELL, C code, shell scripts, UNIFY report writer, RPT.	UNIFY Perl Script	Expansion	63,000	$.43
DISMS	COBOL, Supra/MANTIS DBMS.	SLAP TSO/ISPF, MANTIS full screen editor, FileAid, ISPF	Procedural	9,177 LOC 20 COBOL, 8 Mantis programs	$5.05
MOCAS Production Redesign	COBOL, MANTIS	SLAP Mandate2 MandateX	Procedural	124,313 LOC, 61on-line, 51 batch programs	$.63
SAMMS Financial Funds History	COBOL	SLAP Mandate	Procedural and Expansion	104,061	$1.06
Technology - Automated Password Change Facility	TSO/ISPF workbench for MVS editing. VI editor for UNIX shell, C modifications. In-house-developed C code scanner	SLAP, MANDATE X, MANDATE C	Procedural	7,843	$2.28
DFAMS AIS-Wide Analysis	COBOL, ALC, a proprietary API, Communications software. Data-Unify, Oracle, and CA-Datacom databases.	Specially developed search tool for the M204 code; SLAP for COBOL	N/A	2,016,672	NA

projects/year2k/year2k.html). The process was based in part on a premise that different situations could call for different certification levels reflecting various degrees of risk. However, after presentation to and feedback from the CCB, we revised the certification documents again, and the idea of multiple certification levels was ultimately dropped.

Certification Process

The process covers internally developed configuration items only. A separate initiative addresses the compliance of vendor-provided software and hardware products. Our deployment environment is under the control of another DoD agency; therefore, our participation in the compliance of the deployment environment is limited to the communication of issues.

The Y2K Program office provides certification guidance and a checklist. The system support group returns the completed checklist and test documentation. The program office verifies incomplete or inconsistent information and recommends a certification action prior to presentation to the customer for final certification.

Lessons Learned

- In spite of our efforts to streamline it, the checklist is lengthy, and although the majority of the questions are "yes or no," determining the answers, unless the process is to be shortchanged, requires a fair amount of testing. System proponents that believe their systems are already compliant tend not to plan resources for Y2K testing and certification. Convincing system support groups and funding proponents of the necessity to certify Y2K compliance has been, and continues to be, difficult.
- Industry guidelines or cost models existed for multiphase conversion efforts but did not for certification-only efforts; thus, further hampering attempts to encourage Y2K effort and cost planning for systems already maintained to be compliant.
- In spite of our best efforts to baseline and keep the certification process stable throughout the duration of the program, it is likely going to change as we evolve the process and as higher levels within DoD become increasingly involved in providing Y2K guidance. This compelling interest from higher levels of our organization has at times caused us to question our early formed tenet that moving ahead without such guidance was preferable to waiting for guidance that might come too late to be of benefit.
- The process was designed to provide Y2K test information without being onerous to the projects. We encouraged the use of established software engineering testing and certification processes to the greatest extent possible. However, in working this cross-organizational initiative, we have found test process inadequacies and inconsistencies across the organization. A common benchmark, such as an organization-wide certification process, is difficult to achieve without the existence of common processes.

Pilot Project Study

The Y2K Program proposed pilot projects as a vehicle to discover early Y2K lessons learned relevant to the Y2K change effort. Three purposes would be served:

- Generation of actual conversion cost metrics.
- Identification of not-yet-anticipated Y2K challenges.
- Jump-start of the conversion effort through these early pilots.

Pilot Project Profile

Table 2 summarizes the characteristics of six Y2K conversion pilot projects.

Lessons Learned

Estimation of the Effort Involved. Of immediate note is the variance in pilot project lines of code (LOC) costs. For smaller (less than 10,000 LOC) samples, the LOC cost was higher. We also found that the lack of a date simulation tool adversely impacted the testing, causing extra delays and a reduction in the quality of testing.

Renovation. The selected approach for making a system Y2K compliant depends on the specific AIS and its design, interfaces, and implementation environment. Although smaller, more modern systems are able to successfully perform date expansion, larger legacy systems with multiple interfaces would require considerable more time and effort to do so, for both modification and testing. Expanding the date data field enables the meeting of standards but may cause considerable down time during implementation—partially because of the lessened ability to incrementally implement converted segments—and may cause cosmetic display challenges. In some cases, a procedural approach is the only risk viable approach to achieving Y2K compliance in time.

Awareness and Communications. A system can be Y2K-compliant without conversion of the user-interface layer, i.e., report and screen displays, the modification of which could increase remediation costs and schedules unacceptably. In one of the pilots, we found that the number of on-line programs that needed to be changed increased from six to 60 with the remediation of the user-interface layer. It is important for the user to understand the impacts of various system, interface, and user interface renovation choices.

Customers and users also need to know that delays in replacement system implementation increases the Y2K failure risk to the existing system. Additionally, intention to replace a legacy system does not entirely relieve the customer of Y2K-related costs; contract modifications and additional testing and certification activities may still be required to produce Y2K-compliant new systems.

At the time of the pilots, effective Y2K communication within DSDC was limited; therefore, tool and technology capabilities used by one pilot were not effectively used by *all* pilots. To compensate for this lack of communication, the program office planned and conducted a series of sessions for projects to share ideas, concerns, and information.

Tools. In-house tool capability was assessed as being sufficient for mainframe-based application assessment and renovation phases. Tools for other platforms are becoming increasingly available. However, problems related to the lack of a robust test environment and date simulation utilities caused delays during pilot testing efforts. This realization from pilot efforts escalated our tool assessment and procurement initiatives.

Testing. Testing time and effort was underestimated for all pilots. In one pilot, test processes were lacking; for example, system and functional test plans were developed but lacked the specificity required to sufficiently address the Y2K concerns. The addition of tools and training, without effective test processes, would probably not be sufficient to ensure success of the effort. Consequently, we incorporated testing expertise to guide the test plan development and implementation, but the responsibility for the planning and implementation remained at AIS level. Each AIS was encouraged to develop a testing strategy for specific needs related to test environments, test tools, test utilities, and test data. The program office provides support in tool procurement, installation, and training.

Release Management. Close coordination with deployment environments is indicated not only to ensure adequate support for the installation of support tool releases but also for a larger-than-normal number of new application releases into the field. Where file conversions are involved, coordination is particularly important as other nonstandard DLA systems use file formats passed by our standard systems.

Observations and Conclusions

Throughout this article, we have discussed our approaches, results, and lessons learned as they pertain to significant aspects of our effort. Here we offer some overall observations and conclusions.

Provide a Sufficient Resource Investment

Resourcing Y2K initiatives is not a one-time commitment because resourcing will continuously be a challenge at all levels of Y2K execution. In the current DoD environment, a key strategy for dealing with resource shortages is contracting. But even our contractors are having difficulty retaining enough people to execute their contracts in a tight, competitive labor market. Another challenge we have had in the government is in keeping the noncontracted, what we call organic personnel commitments stable. We are constantly downsizing—even offering incentives for early retirement—and no resource commitment is a sure bet.

Provide Sufficient Investment in Testing

Every pilot incurred slowdowns and obstacles during the testing phase. As discussed, we discovered that variances in testing procedures within the organization affected attempts to baseline a certification process. Effort expended in defining Y2K-specific testing practices and in building adequate test environments are worth the front-end investment.

Be Prepared for High-Level Visibility

We recognize the benefit of high visibility as the crucial components of sponsorship, and commitment can wax and wane without it. However, visibility can have negative impact. We quickly became swamped with project and progress requests for information from various government agencies. In hindsight, designation of a communications officer would have been invaluable in answering these calls for information. Lack of such a resource affected our ability to stay focused on developing strategy and plans and on implementation to the desired degree.

Summary

Following is a recapitulation of what we have found to be the most critical factors affecting our Y2K remediation effort:

- Sustaining organizational commitment.
- Understanding the organizational baseline at the outset of the initiative.
- Understanding the challenge in applying standard remediation solutions across an organization with disparate processes.
- The adequacy of testing tools *and* processes.
- Focus on the underlying hardware and system software is a major component of the total picture. We did not anticipate the level of work and difficulty involved in this aspect at the outset.
- Obviously, remediation for systems with future date projections should be a first priority. But there must also be prioritization and contingency planning *within* the system effort, because failures will surely occur even as the system is being renovated.
- Carving out enough time for integrated testing of renovated applications with upgraded system software and hardware may turn out to be our biggest challenge. ◆

About the Author

Sarah J. Reed began her DoD career with the DSDC in 1987. In that time, she has been involved in or led a number of projects, such as corporate-wide metrics collection, analysis and implementation, the Organizational Culture Initiative, and fee-for-service implementation.

In 1991, she worked with the DLA Information Technology Policy Board representative to review, suggest, and comment on DoD policy, some of which shapes the DoD mission today. In 1992, she led the effort to establish a baseline Capability Maturity Model assessment for the organization. In December 1995, she accepted the challenge to lead the Y2K Program at DSDC. In December 1997, she became a private consultant on Y2K issues.

DSDC's Year 2000 Program office is now being managed by David Koppy. For further information on DSDC's Year 2000 Program please, contact

Cindy Nickoson
DLA Systems Design Center
Columbus, OH 43213
Voice: 614-692-9296 DSN 850-9296
Fax: 614-692-8393 DSN 850-8393
E-mail: cnickoson@dsdc.dla.mil

If you have questions for the author, please contact

Sarah Reed
CompuWare
Voice: 614-847-8212

Software Inspections and the Year 2000 Problem

Don O'Neill
Independent Consultant

The year 2000 (Y2K) problem promises to impact information systems of all kinds. With no silver bullet available, the responsible manager must take a variety of actions to detect and correct this problem. This article provides a compliance checklist to aid in Y2K problem detection and correction.

Software inspections are a powerful mechanism to help detect and correct the Y2K problem. Software inspections call for a close and strict examination of software artifacts against the standard of excellence set by the organization, significantly improving defect detection and removal capabilities. By explicitly setting the standard of excellence for software products to operate correctly before, during, and after 2000, and by conducting software inspections on software products thought to be date sensitive, the responsible manager is taking an important and prudent step to meet the challenge.

To reason effectively about Y2K compliance at the program level, the reviewer must understand the standard date format and treatment by system date routines, identify and verify internal date usage, identify and verify external file date usage, and understand the context of date usage within the application domain. Specifically:

- Reasoning about internal date usage includes verifying date representations, evaluating date transformations and logical expressions, identifying variable names that contain date types, and recognizing and handling the leap year anomaly. Not all problems lie in looking forward; looking-backward calculations must also be identified and assessed.
- Reasoning about external file date usage includes identifying external system interfaces and verifying standard date format for input and output date records.
- Reasoning about the context of date usage within the application domain requires an understanding of the time horizon to failure (THF). For example, the THF for a 30-year mortgage is 1970; for an enterprise five-year plan, 1995; for a four-year motor vehicle license, 1996; for a two-year credit card issuance, 1998; and for various annual deadlines, 1999.

To help practitioners uncover all possible Y2K problem situations, the following Y2K compliance checklist is added to the standard of excellence set by the organization. This checklist is drawn from the software inspections course and lab I offer. These checklists are organized along a common framework that includes completeness, cor-

Year 2000 Compliance Checklist

1. Has the product component been assessed for Y2K compliance?
1.1 Is Y2K compliance specified as standard date format (such as, YYYYMMDD)?
1.2 Are date-related system services identified and Y2K compliant?
1.2.1 Are system date routines identified and Y2K compliant?
1.3 Are all date-related nodes and flow graphs identified and Y2K compliant?
1.3.1 Is internal date usage identified and Y2K compliant?
1.3.1.1 Are all date representations identified and Y2K compliant?
1.3.1.2 Are all date transformation routines identified and Y2K compliant?
1.3.1.3 Are all date-related names identified and Y2K compliant?
1.3.1.4 Are all date calculations identified and Y2K compliant?
1.3.1.5 Are all date uses in logical expressions identified and Y2K compliant?
1.3.1.6 Are all leap year computations identified and Y2K compliant?
1.3.1.7 Are all "looking backward" (from the year 2000) calculations identified and assessed as Y2K compliant?
1.3.1.8 Are all "looking forward" (from the year 2000) calculations identified and assessed as Y2K compliant?
1.3.2 Are all files identified and Y2K compliant?
1.3.2.1 Are all external system interfaces identified and Y2K compliant?
1.3.2.2 Are all input and output date records formatted as standard date format (such as, YYYYMMDD) and Y2K compliant?
1.3.2.3 Are all extended semantics (embedded dates and sort keys) identified and Y2K compliant?
1.3.2.4 Are all imported files identified and Y2K compliant?
1.3.3 Has the Year 2000 Time Horizon to Failure (THF) been identified for the application?
1.3.3.1 What is the THF for the application?

rectness, style, rules of construction, multiple views, technology, and metrics. The Year 2000 Compliance Checklist is one of the multiple views for design and code. For more information on this software inspections training and the results it obtains, please visit http://members.aol.com/ONeillDon/index.html. ◆

About the Author

Don O'Neill is an experienced software engineering manager and technologist currently serving as an independent consultant. Following 27 years with IBM's Federal Systems Division, he completed a three-year residency at Carnegie Mellon University's Software Engineering Institute under IBM's Technical Academic Career Program. As an independent consultant, O'Neill conducts defined programs to manage strategic software improvement, including an organizational software inspections process, directing the National Software Quality Experiment, implementing software risk management on the project, conducting the Project Suite Key Process Area Defined Program, and conducting Global Software Competitiveness assessments. He served on the executive board of the Institute of Electrical and Electronics Engineers (IEEE) Software Engineering Technical Committee and as a distinguished visitor of the IEEE. He is a founding member of the National Software Council and the Washington, D.C. Software Process Improvement Network.

9305 Kobe Way
Gaithersburg, MD 20879
Voice: 301-990-0377
Fax: 301-670-0234
E-mail: ONeillDon@aol.com
Internet: http://members.aol.com/ONeillDon/index.html

Coming Events

Ownership and Control Issues in Architecture-Based Acquisition of Product Lines

Dates: Jan. 13-14, 1998
Location: Pittsburgh, Pa.
Subject: The objective of this meeting is to produce a short point paper articulating the acquisition business models identified at the Salem '97 workshop.
Sponsor: Software Engineering Institute
Contact: James Withey
E-mail: jvw@sei.cmu.eu

Configuration Management Seminars: (1) Basic, (2) Advanced, (3) Comprehensive

Locations: Bethesda, Md.; San Diego, Calif.; Washington, D.C.; Las Vegas, Nev.
Dates: Jan. 26 – March 11. Contact Dana Marcus for specific date of each seminar.
Subjects: (1) Basic configuration management (CM) course for individuals in the configuration field for six months or longer. The latest CM standards and requirements; scope and elements of a good CM plan; English release requirements; establishing appropriate baselines; managing CM status accounting records and reports; guidelines for handling and documenting variances, etc. (2) Advanced CM course for individuals possessing the basic knowledge of the CM field; impact of COTS, NOTS, and NDI on CM requirements for Department of Defense procurement; developing models and metrics for CM products and processes; establishing comprehensive change management and corrective action systems. (3) A comprehensive methodology for implementing CM; incorporating the best practices from industry leaders, the latest technology, and the newest standards, guidelines, and requirements, including the new standards J-STD-016, ISO122207, US 122207, EIA-649, and MIL-HNBK-61.
Sponsor: Technology Training Corporation
Contact: Dana Marcus
Voice: 310-534-3922
E-mail: dmarcus@ttcus.com

First Workshop on Biologically Inspired Solutions to Parallel Processing Problems (BioSP3)

Dates: March 30 – April 3, 1998
Location: Orlando, Fla.
Subject: This workshop seeks to provide an opportunity for researchers to explore the connection between biologically based techniques and the development of solutions to problems that arise in parallel processing.
Sponsor: IEEE Technical Committee on Parallel Processing (tentative)
Contact: http://www.ee.uwa.edu.au/staff/zomaya.a.html/BioSP3.html

IEEE Computer Society International Conference on Computer Languages 1998

Dates: May 14-16, 1998
Location: Loyola University Chicago, Chicago, Ill.
Sponsor: IEEE Computer Society, Technical Committee on Computer Languages in cooperation with the Association for Computing Machinery Special Interest Group on Programming Languages.
Contact: http://www.math.luc.edu/iccl98/

Certifying Year 2000 "Fixes"

Jeffrey Voas
Reliable Software Technologies

There is much less talk about certifying the correctness of year 2000 (Y2K) conversions than there is about how to make the conversions. Certifying that Y2K "fixes" were appropriate can be done easily by using a combination of different software testing techniques. This article describes these techniques and why they should be considered essential processes in any Y2K conversion solution.

The lifecycle cost to maintain software can exceed the costs to develop the original code. This is particularly true for software systems that are expected to continue in service for 20 to 30 years, in which maintenance can account for 50 percent to 70 percent of the total lifecycle costs for a software system. Of these maintenance costs, testing can account for 50 percent or more of the costs [1]. Thus, the cost to test modified code can be a substantial portion of the total costs of keeping legacy code alive.

These factors hold true for Y2K fixes. To eliminate a system's Y2K problem requires that you identify what parts of the software need modification, make the conversions, and test the conversions. According to [4], more than 60 percent of all Y2K costs will go to testing. This figure can be as high as 70 percent for some projects [5].

In a regular system's lifecycle, the maintenance phase involves two key activities: fixing faults in existing code and adding new functionality to existing code. Y2K conversions can be placed into either category, depending on your perspective and the age of the system. If you view Y2K problems as fixing programmer mistakes, your fixes would be considered fault eradication. (This makes sense if the systems were built fairly recently.) If you view the Y2K problem as software that outlived its intended lifespan, you view Y2K conversions as adding functionality to aged systems. (This makes sense for systems that incorporate code created many years ago.)

Regardless of how the Y2K issue is viewed, modified code should be tested, and unmodified parts of the system should be retested to ensure that each "fixed" system is Y2K immune. As already stated, these testing expenses can be pricey. One reason is that few testing tools are smart enough to automatically know how to minimize testing costs for modified code. However, you would think a "smart" tool could determine exactly what code needed to be retested. It would be great if such an automated tool existed to distinguish that kind of code in a "optimized" mode, i.e., determine the least amount of code that needed to be retested to demonstrate that a code conversion was correct.

Unfortunately, no such tool exists. This suggests that there is a serious need for tools that seamlessly integrate with Y2K conversion tools and that test Y2K conversions. If such tools existed, the total global cost of the Y2K problem could be reduced while still providing sufficient confidence that Y2K conversions were correct. This could add up to astronomical savings, as the world-wide cost for fixes alone is $600 billion, not to mention legal liability costs that could exceed $1 trillion [2, 3].

Following is a description of what testing tools must accomplish under any scenario to provide the appropriate levels of confidence. In short, it is something like a checklist of testing processes that certify Y2K compliance.

Coverage Testing

The absolute minimal requirement is that modified code be tested (commonly referred to as "exercised"). If modified code is not exercised, it is not possible to know what its behavior will be. To exercise code, generate test cases that execute the conversions, then employ simple coverage analysis to analyze whether modified statements are hit. This can be done easily if

- The conversion tool places comments in the converted code.
- The coverage tool's parser looks for those comments.
- The coverage tool then places instrumentation to record when those statements are exercised.

Once coverage testing is successfully completed, you know that all code modifications have been executed at least once.

But from a quality perspective, it is imperative to recognize that it is barely sufficient to merely reach statements, because there are other forms of coverage testing, e.g., dataflow, that are better at fault detection than statement testing. For example, dataflow testing would allow you to test "all uses" of the year fields that were modified. Or if you attacked your Y2K problem by adding complex conditions, e.g., changing

```
if y1 < y2 then
   years_apart = |y2 - y1|
```

to

```
if (((y1 < y2) and (y2 < 00)) or ((y1 >= 00) and (y2 >= 00))) then
   years_apart = |y2 - y1|
else if (y1 <= 99) and (y2 >= 00) then
   years_apart = (99 - y1 + 1) + (y2 - 00)
```

to avoid increasing the size of year fields, you should use a coverage testing approach like multiple-condition coverage

(MCC) or condition-decision coverage (C/DC). Note that dataflow, MCC, and C/DC coverage testing are more thorough than statement testing.

However, coverage testing is only one ingredient in Y2K certification. After all, even complete coverage testing of all code modifications does not imply that all Y2K conversions are correct. All fixes could be correct, but the fixes may have broken system functionality; that is, all the year fields may now work properly, but other functionality that used to work now does not. (Such a situation could occur because of a lurking fault that could not be triggered until a year greater than 1999 is used.)

This potential problem needs to be mitigated by retesting existing functionality. To do so, regression testing can be employed, which is the next ingredient needed for Y2K certification.

Regression Testing

Regression testing employs a suite of test cases (usually with respect to the requirements or specification) to ensure the outputs from the original code and converted code are identical for each member of the suite. Note here that it is assumed that for each member of the suite, there should be identical behavior for both versions of the code. Regression testing provides evidence that the conversions have not affected any functionality that should have remained unaffected.

System-Level Testing

But there will also be system-level inputs that we would want to result in different outputs (between the converted and unconverted programs). If this were not the case, why was the software converted? To determine that the new version is doing what you want for these inputs, employ system-level testing, which will employ test cases that represent events beyond 1999. System-level testing will serve as our last ingredient in Y2K certification.

It is important to note that system-level testing is neither a substitute for coverage testing nor does coverage testing replace system-level testing. It is possible to exercise all code conversions and not discover that a conversion fails in the context of a test that represents an event after 1999. Likewise, it is possible to system-level test with a wide variety of post-1999 scenarios that do not exercise all modifications. Thus, both forms of testing are needed for Y2K certification.

Summary

To certify that code is Y2K compliant, three different forms of testing should be employed:

- Coverage testing to exercise fixes.
- Regression testing to see whether new code breaks other system requirements that are not related to calendar dates.
- System-level test to see how the new system handles events past 1999.

These techniques do not guarantee that the conversions will integrate into your system and work correctly under all scenarios. Except for exhaustive testing, testing can never make such guarantees. Instead, testing provides confidence. And for legacy Y2K systems, that is what is needed after these "tried-and-true" systems are upgraded to handle events after 1999.

Since testing can account for 50 percent of maintenance costs, if you were planning to spend X dollars on conversion, the additional certification costs could double your cost to 2X. But without taking these defensive measures, you could get fooled into thinking that your Y2K problem is behind you when it is not. Given that you have taken proactive measures to solve your Y2K problem, this is a situation you will want to avoid.

I have deliberately simplified the Y2K certification process down to a handful of traditional testing approaches. Admittedly, there are more advanced certification processes that could be employed to provide similar results. Given that much Y2K conversion is ongoing without assurances that the conversions are correct, it should be more beneficial for practitioners to layout the basic needs for Y2K certification than to layout a Utopian "pipe dream," such as proving beyond all doubt that the legacy system is correct. ◆

About the Author

Jeffrey Voas is a co-founder of and chief scientist for Reliable Software Technologies and is currently the principal investigator on research initiatives for the Defense Advanced Research Projects Agency and the National Institute of Standards and Technology. He has published over 85 refereed journal and conference papers. He co-wrote *Software Assessment: Reliability, Safety, Testability* (John Wiley & Sons, 1995) and *Software Fault-Injection: Inoculating Programs Against Errors* (John Wiley & Sons, 1997). His current research interests include information security metrics, software dependability metrics, software liability and certification, software safety and testing, and information warfare tactics. He is a member of the Institute of Electrical and Electronics Engineers and he holds a doctorate in computer science from the College of William & Mary.

Reliable Software Technologies
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166
Voice: 703-404-9293
Fax: 703-404-9295
E-mail: jmvoas@rstcorp.com

References

1. Myers, G., *The Art of Software Testing*, John Wiley & Sons, 1979.
2. "Year 2000 Prophet Preaches $600 Billion Digital Fix," *Computer News Daily*, Oct. 1, 1997.
3. Hassett, D., "Frequently Asked Questions About the Year 2000 Problem," available at http://www.y2k.com/legalfaq.htm.
4. Tech-Beamers, "White Paper Year 2000 Focus On Testing," May 10, 1996, available at http://www1.mhv.net/~techbmrs/tstgwp.htm.
5. Scheier, R. L., "Year 2000: Testing Can't Wait," *Computerworld*, Oct. 20, 1997, available at http://www2.computerworld.com/home/online9697.nsf/All/971020test.

Year 2000 Web Sites

These Internet sites show how the defense software community is tackling the year 2000 problem.

Air Force Materiel Command (AFMC) Year 2000 Bomb Squad Unit

http://www.cisf.af.mil/seit/seit/progmgmt/y2kBSU.htm

- Compliance strategy (ITW/AA).
- Program management plan (ITW/AA).
- Compliance checklist.
- Working group.
- Overview briefing.
- Test plan.
- Test schedule.
- Verification cross-reference matrix.

U.S. Air Force Year 2000 Page

http://year2000.af.mil

- Year 2000 infrastructure guidance package.
- Spreadsheets.
- Automated information systems certification and testing.
- Certifier training slides.
- Certification-tracking documents.
- Compliance checklist.
- Product certification information and resources.
- Briefings.
- Working group minutes.
- Newsletter.
- Best practices.

Software Technology Support Center (STSC)

http://www.stsc.hill.af.mil/RENG/index.html#2000

- Articles.
- Database of commercial year 2000 tools and services.
- The year 2000 tools and services lists can be downloaded on this page or customized lists are available.
- Year 2000 tool evaluation reports.
- Provides year 2000 services to other Air Force (and some government) organizations.
- Links to other year 2000 pages.

Defense Information Systems Agency (DISA) Office of Chief Information Officer (CIO)

http://www.dsdc.dla.mil/priv/projects/year2k/year2k.html

- Year 2000 awareness.
- Articles.
- Presentations.
- Conference information.
- Status of DISA's defense megacenters.
- Compliance, test, and evaluation information.

The Year 2000 Information Center

http://www.year2000.com/cgi-bin/y2k/year2000.cgi

- Articles.
- User groups.
- Vendor links.
- Products to build awareness.

Army Year 2000 Page

http://www.army.mil/army-y2k/

- Project change of century action plan.
- Briefings.
- General information (articles, etc.).

U.S. Army Year 2000 Technical Integration Center (TIC)

http://www.hqisec.army.mil/y2kweb/

Compliance information on
- Personal computers.
- Personal computer Y2K test results.
- Mini computers and Y2K.
- Network equipment.
- Commercial software.
- TIC Y2K capabilities.

Defense Logistics Agency

http://www.dsdc.dla.mil/priv/projects/year2k/year2k.html

- Briefings.
- Certification guidance and checklist.
- Process and tools.
- Year 2000 issues.
- Reports.

Office of Information Technology Year 2000 Information Directory
CIO Council Subcommittee on Year 2000

http://www.itpolicy.gsa.gov/mks/yr2000/y201toc1.htm

- Conferences.
- Best practices.
- White papers and articles.
- International year 2000 links.

Federal Aviation Administration Year 2000

http://www.faa.gov/ait/year2000/2000home.htm

- Year 2000 guidance.
- Year 2000 presentation.
- Compliance criteria.

A Good Idea Still Requires a Good Implementation

David P. Quinn
National Security Agency

No matter how good an idea may be, a poor implementation may lead people to believe it is a bad idea. The Capability Maturity Model (CMM)^{SM} is no exception. Many organizations have not had success implementing the CMM and believe it to be a bad idea. To help organizations in their implementation of the CMM, this article presents different implementation approaches and the organizational situations where they are most appropriate.

Warren Keuffel recently criticized the CMM in "Coding Cowboys and Software Processes" (CROSSTALK, August 1997). He spoke of his unsuccessful implementation of the CMM as part of a Software Engineering Process Group (SEPG). On the basis of this failed attempt, he assumes that the model is flawed and is a bad idea.

This is a common problem that I see while working with software organizations. Despite success that numerous organizations have had with the CMM, many believe the model is a bad idea because it did not work for them. They need to step back and look at whether the fault lies with the model or with their method of implementation.

There are a variety of ways to define and deploy usable processes in software organizations. The challenge is to find the right combination of definition and deployment for your organization. This article discusses ways to define and deploy usable processes to build an organization standard software process (OSSP).

Defining Usable Processes

The appropriate way to define usable processes depends on the organization's culture or situation. Each method below includes examples of appropriate organizational culture or situations.

Process for Hire

One way to define usable processes is to buy the processes from a vendor, much like buying a set of encyclopedias. The organization can buy the entire set or buy the processes one volume (one key process area) at a time. Of course when you buy the process, you will need to buy additional services like training, templates, and other associated assets. The organization's improvement investment is primarily financial.

Although many mature organizations warn that you must grow your own processes, Process for Hire can be successfully implemented in two situations. The first is when a new organization has just been formed; there is no culture to fight at this point, and people are likely to welcome a defined direction. The second circumstance is when the organization is given a mandate to achieve a certain maturity level by a pre-defined time or lose a critical customer or financing. The organization must change quickly or it will disappear.

No matter which circumstance the organization faces, the organization must break the purchased processes down, make alterations for their particular environment, and put the processes back together. This procedure ensures that the process will work in the given environment and that the organization takes ownership of the process.

The Ivory Tower

Also known as the "Ten Commandments" approach, this method isolates a handful of experts in a workspace to develop all the processes for the organization to follow. When they are finished, they bring the processes down from their "tower"—possibly written on stone tablets for effect—and expect the organization to follow them as written. They impart their collective experience to the organization and expect it to be accepted solely on faith.

This method works well when the organization has a large number of junior engineers who lack the experience to

Table 1. *Methods of definition.*

Method	Characteristic	Culture/Situation
Process for Hire	Using another organization's processes.	Start-up organizations. Process improvement is mandated.
Ivory Tower	Team of senior experts.	Informal culture. Junior workforce.
Process Action Teams	Hands-on expertise.	Long-term process improvement approach.
Bubble Up	Working processes submitted.	Strong process improvement resistance.
Self-Selection	Processes are nominated.	Pockets of good process exist.
Shadowing	SEPG follows projects and documents them.	Too busy to improve.

Capability Maturity Model and CMM are service marks of Carnegie Mellon University.

develop usable processes. It is especially effective in an organization that prides itself on its informal atmosphere. By selecting senior experts to define the processes, the organization uses the people who probably forged that informal culture and can best alter it to a more mature environment.

However, the organization must be cautious when using this method, especially when the senior experts have not been technically involved in software development for some time. The "experts" could develop processes that do not match the current working environment or culture. This method therefore requires constant communication with the workers to ensure they understand how the process was developed.

Process Action Teams

The Ivory Tower method should not be confused with the use of Process Action Teams (PATs). PATs focus on one process area or one thread of software engineering that covers multiple process areas, whereas the Ivory Tower approach addresses the entire process improvement effort. PATs are composed of hands-on technicians with up-to-date expertise in the area being defined. They define the process based on current practices, then create additional processes that fill specific CMM key process area requirements.

This is the most common approach to define processes, and is usually used to implement improvements incrementally in organizations that view process improvement as a long-term investment. It requires the SEPG to handle the institutionalization aspects and track the interfaces between process areas.

Bubble Up

Nothing breeds success like success. This method requires SEPG members to document how successful projects operate, then use those processes to form the OSSP. The key to making this method work is to define the environment in which the process was successful.

This method works well in organizations that are resistant to process improvement, i.e., project teams that will likely fight processes that have not been used in practice. The knowledge that successful projects have used the process makes buy-in more likely. However, these processes may be incomplete according to the CMM and may need adjustment in the future. Unfortunately, this method focuses on creating documented processes, complete or not.

The challenge comes in how to define a successful project. The organization cannot only look at *whether* the project achieved its goals, but *how* it achieved them. Before implementing this method, the organization must define a successful project in terms of its business goals and engineering needs.

Self-Selection

This method requires that project members submit processes they believe are worthy for the organization to use. They know that the processes work and believe that other projects could successfully use them. Management also may draw on these people's experiences and submit processes they believe other projects could use.

This method is especially useful when groups have independently established mature practices, without waiting for the rest of the organization to improve. This method also speeds up the adoption process, since the processes are known to work in the organization. The SEPG's job is to determine whether the processes are complete or need to be expanded to cover CMM practices.

The self-selection method can cause problems if projects view it as a form of self-congratulation. As with the Bubble Up method, self-selection requires that acceptance criteria be in place to ensure the submitted processes are truly worthy for use across the organization.

Shadowing

Project workers do not always have the time or ability to document how they perform their processes. Shadowing helps this type of project. The SEPG follows the project and documents its processes. Project members then review what the SEPG has written to confirm that the description reflects their processes. The project now has its processes documented, and the SEPG has a candidate process for its OSSP.

In addition to creating documented working processes, this method supports the waiver process needed for process improvement. If a particular project needs to use processes outside the norm, or if project members merely want to try something different, the SEPG can shadow the project to document the differences and determine whether they are improvements.

This method best fits an organization that resists process improvement or claims to be too busy to improve. With a minimal investment, the process can be documented, and once documented, it can be improved based on the CMM process descriptions.

Deploying Usable Processes

Defining usable processes is only half the fun. Once they are defined and determined to be good enough for other projects, the SEPG must find ways to deploy them. This again depends on the current organizational culture and situation.

Starting Monday

This method relies on a mandate that the entire organization begin to use the same process on a certain day or date, usually a Monday or the first of the

Table 2. *Methods of deployment.*

Method	Characteristic	Culture/Situation
Starting Monday	Mandated start date.	Immediate conformance.
Test the Water	Pilot testing process.	Proven processes before immersion.
Fill in the Gaps	Provide missing practices.	Too busy or process improvement resistance.

Web Addition

This article can be found in its entirety on the Software Technology Support Center Web site at http://www.stsc.hill.af.mil/CrossTalk/crostalk.html. Go to the "Web Addition" section of the table of contents.

Achieving Information Superiority

Lt. Gen. Ronald T. Kadish
Commander, Electronic Systems Center, Hanscom Air Force Base, Mass.

In a speech given at the Air Force Information Technology Conference in September 1997 at Standard Systems Group, Montgomery, Ala., Lt. Gen. Kadish describes the new framework to be employed by Air Force in its quest to achieve information superiority. Included in the strategy are an acquisition cycle time of 18 months, seamless connections between classified and unclassified systems, and commercial-off-the-shelf applications that are open system, based on standards.

month. There is no transition period and no choice. The organization must begin to use the process on exactly that date—or else.

This method is commonly used when there is an extreme need for conformity. For instance, if the organization is building safety-critical systems, it cannot continue to use a flawed process. Or perhaps the organization may be attempting to re-baseline measures in its process database. The data must be consistent, so conformance to the process is mandated.

With a short suspense date for implementation, the SEPG must quickly train the technical staff to be successful. They must be prepared for a tremendous number of trouble calls for help with the process. Automation may lag behind but should eventually catch up.

Testing the Water

Processes do not always work right the first time, so they are tested through the use of pilots—much like testing the water with your toe. The process is tried in selected areas to see if it is good enough for organization-wide use. If the water feels right and the process works, the entire organization will jump into the water.

This is the most common deployment method for process improvement. Most organizations want to make sure a change will be an improvement before exposing the entire organization to it. This builds confidence in the process and fosters automation of the process.

The SEPG should also train the organization in any new method in several training sessions for different groups. The overall amount of training is based on whether the rest of the organization is going to jump into the water at once or ease into the water one part at a time.

Fill in the Gaps

Projects usually have practices in place that, for the most part, are successful; members only need to fill in the gaps between what they are doing and what needs to be done. As gaps are found, they are filled based on the organization's defined processes. By making minor adjustments to their current practices, project members incorporate process improvement into their ongoing product development. With time, the project has all the gaps filled and works according to the organization's defined processes.

This method works well when projects claim to be too busy to implement improvement. The changes are incremental and meaningful and based on current practices. The project improves without team members having to concede that they are actively working on process improvement.

The SEPG's job is more complicated in this method because they are constantly doing gap analyses. The relationship with the projects is more detailed and drawn out. Improvement takes longer but is likely to have greater buy-in and be institutionalized without as much verification.

Conclusion

Poor implementation of a good idea does not make it a bad idea. No matter how wonderful an idea is at its inception, its implementation will determine whether it continues to be good. The CMM is no exception. It has been proven to be an excellent tool for improvement, but it must be implemented in a way that matches the organization's culture and present circumstances. Process improvement requires different approaches to process definition and deployment. The organization must find the right combination of definition and deployment to benefit from the CMM. ◆

About the Author

David Quinn is technical director of the Software Engineering Knowledge-Based Center at the National Security Agency. He has over 14 years software development and maintenance experience. He spent the last four years working on software process improvement as a process consultant. He is certified by the Software Engineering Institute as a lead assessor for CMM-Based Appraisals for Internal Process Improvement. He also is a member of the CMM Advisory Board.

National Security Agency
9800 Savage Road, Suite 6639
Ft. Meade, MD 20755-6639
Voice: 301-688-9440
Fax: 301-688-9436
E-mail: dpqsr@romulus.ncsc.mil

The Year 2000 Farce

Peter Errington

We should keep the two-digit year fields when solving year 2000 (Y2K) problems. Convert the existing series of year values to a string of values that the application programs can handle without error. Then convert all year values back before the outputs are produced. This will reduce the Y2K problem to primarily a job control language exercise.

It has been estimated that $600 billion will have to be spent worldwide to fix the Y2K problem that afflicts the computer world. Most of this money will be wasted because of a poor problem definition. It does not take a computer genius to realize this—a smart kid could figure it out. Allow me to illustrate.

SK (Smart Kid) sat on her living room sofa moodily gazing at a copy of the local county newspaper, where she had just read that her county government had estimated that it would have to spend $44 million to fix the Y2K problems for all its computer systems. (Not the whole state of Maryland, just her little county.)

SK was not a "computernick." She used word processing on her computer to write papers faster, but she was totally uninterested in the computer toys that so fascinated some of her classmates. Nevertheless, she indulged herself in a silly daydream based on what she had just read in the newspaper.

In this daydream, Dwight Eisenhower was not just the pretty-good president he was in reality, but had single-handedly ended war and hunger forever. As a result, the world revised the calendar around him. The term AD no longer stood for Anno Domini, but Age of Dwight. Year AD 1 was the year Eisenhower started planning to make his move on the presidency, formerly known as 1951.

Great, thought SK. If we had actually rearranged the calendar, we would not now be worrying about spending $600 billion on the Y2K problem. It then occurred to SK that for the computational purposes of computer programs, we could pretend that the above scenario had indeed taken place. She was unsure of the details, not having computer experience, but she was sure it could be done.

I know it can be done, and I will describe how. First, however, understand that I am only discussing batch-processing computer systems. I know nothing about the problems of elevators, heart pacemakers, and other such devices. Now let us define the real problem for batch-processing computer systems.

A Poor Problem Definition

The four-digit year values we use every day (1998, 1999, 2000 ...) would eliminate the Y2K problem were they already integrated in our computer systems. But they are not so integrated, so we must convert the two-digit year fields in our programs to four-digit fields.

A Good Problem Definition

The two-digit year values we use every day and which are in our databases (98, 99, which lead to 00, 01...) will cause these computer programs to malfunction when we reach the year 2000.

We could convert these two-digit values to another series of continuous two-digit values that would not cause problems with computer processing in the year 2000. The following explanation has phrasing that reflects my background as an IBM mainframer, but it contains a sound general solution.

Leave the application programs alone. From any input file that contains year values, create a temporary file in which year values are converted to a continuous series that the application program can handle, then have this temporary file be the input to the application.

For example, assume a time frame starting at the beginning of 1951. This year would be 01 for the application program, 1968 would be 18, 1997 would be 47, and 2006 would be 56. All the calculations, compares, and sorts that involve a year would work fine (assuming there is no data in this system for a year earlier than 1951, in which case, my example would have to be modified).

Then, the output data sets with year fields have to be produced as temporary files in which the year values are converted back (47 becoming 97, 56 becoming 06, etc.) before the final outputs are produced.

The only work involved here would be to change job control language to produce and handle the temporary data sets, to write the truly infinitesimal programs to convert year values, and to move the interaction with the final output devices from the application programs to the final job steps in which the year values are translated back. This would be easy for people who know what they are doing. It would be much easier than pawing line by line through a Himalayan mountain range of application program coding.

As pointed out above, the calculations, compares, and sorts in the application programs would work perfectly. Therefore, my suggested method would give a high level of confidence that is missing in articles I have been reading on the Y2K problem; in these articles, unforeseen surprises are feared from having to change date field lengths.

Some organizations, already worrying whether they can adapt before 2000, have less time than they realize. For example, a “99” in a year field may not represent a year but instead be a flag. Or information may be entered now that pertains to the next century (a loan today may have an end date in the next century, which would cause the program to reject the information with the message that a loan cannot end before it begins. This has actually happened.)

So in addition to ease of conversion and high confidence level, my suggested approach (because ease equals speed) may allow deadlines to be met that are more stringent than some organizations realize.

I have spent considerable time trying to think of valid counter arguments to the above proposal, and those I have come up with have all been weak. The most valid of the lot is that organizations might be required to furnish other organizations with files that contain four-digit year fields. But I cannot imagine an easier programming job than accepting as input a record with a two-position year field(s) and producing an output record with additional characters “19” or “20” added where appropriate.

To summarize, it seems the world is bent on squandering untold billions for no valid reason. ◆

About the Author

Peter Errington has spent his entire career in data processing, starting in 1961. He retired in 1996, having worked for three private firms and two government agencies. From 1961 to 1971, he was employed at the Southern New England Telephone Co., General Electric, and Informatics. In 1971, he joined the Agency for International Development, where he specialized in payroll and personnel systems and economic and social data banks. In 1989, he moved to the Defense Logistics Agency, where, among other things, he worked extensively on Continuous Acquisition and Lifecycle Support and warehouse automation.

3809 Archer Place
Kensington, MD 20895
Voice: 301-946-7232
Fax: 703-767-6564
E-mail C/O: bette_hill@hq.dla.mil

Year 2000 PROGRESS
Mission-Critical Systems of Federal Departments and Agencies

	Assessment Completed	Renovation Completed	Testing Completed	Implementation Any	Rating
SSA Social Security Administration	YES	78%	67%	YES	A-
GSA General Services Administration	YES	35%	26%	YES	B
NSF National Science Foundation	YES	33%	25%	NO	B
SBA Small Business Administration	YES	35%	35%	YES	B
HHS Department of Health and Human Services	YES	28%	10%	YES	B-
EPA Environmental Protection Agency	NO	33%	28%	YES	C
FEMA Federal Emergency Management Agency	NO	35%	35%	YES	C
HUD Department of Housing and Urban Development	YES	9%	2%	YES	C
Interior Department of the Interior	YES	43%	0%	NO	C
Labor Department of Labor	YES	15%	11%	YES	C
State Department of State	YES	25%	0%	NO	C
VA Department of Veterans Affairs	NO	51%	28%	YES	C
DOD Department of Defense	NO	40%	34%	YES	C-
Commerce Department of Commerce	NO	15%	6%	YES	D
DOE Department of Energy	NO	10%	10%	YES	D
Justice Department of Justice	YES	1%	1%	NO	D
NRC Nuclear Regulatory Commission	YES	0%	0%	NO	D
OPM Office of Personnel Management	YES	3%	0%	NO	D
Agriculture Department of Agriculture	NO	8%	4%	YES	D-
NASA National Aeronautics and Space Administration	NO	8%	7%	YES	D-
Treasury Department of the Treasury	NO	6%	5%	YES	D-
AID Agency for International Development	NO	N/A	N/A	N/A	F
DOT Department of Transportation	NO	0%	0%	NO	F
Education Department of Education	NO	0%	0%	NO	F

Prepared for Subcommittee Chairman Steven Horn.
The departments and agencies are responsible for the accuracy and consistency of percentages reported.
Created: Sep. 15, 1997
Revised: N/A
Subcommittee on Government Management, Information, and Technology
Source: http://www.house.gov/reform/y2k/970915score.htm

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.

The Software Technology Support Center Reengineering and Year 2000 Services

Essential Process Improvement

The Software Technology Support Center (STSC) Reengineering and Year 2000 (Y2K) staff has introduced the concept of "Essential Process Improvement" to focus on surviving the Y2K challenge.

Beating the Year 2000 Event Horizon

Your application's event horizon (or expected failure date due to problems with processing date values close to the year 2000) is fast approaching. You may only be able to improve those processes that are absolutely essential to the success of your Y2K upgrades. Other process improvement initiatives may need to be postponed. You may be dealing with poorly documented legacy systems that were built with less than the best practices. The time is limited to redevelop a Y2K-compliant system using improved development methods and tools. What should you do?

Corporate and Project Year 2000 Compliance Guidebooks

Our approach to help organizations conduct each phase of their Y2K compliance project is based on industry best practices and our experience with Y2K upgrades. Corporate or director-level issues that require high-level planning and support need to be identified. Specific project-level issues also need to be addressed. We help an organization tailor our *STSC Corporate Y2K Compliance Guidebook* and our *STSC Y2K Project Guidebook* for their project. These guidebooks work in harmony with the Air Force Communications Agency's guidelines and checklists such as the *Y2K Concept of Operations* and the *Air Force Y2K Compliance Checklist*.

STSC's guidebooks provide a checklist of activities accompanied by supporting rationale to explain the purpose of each activity. As each activity is accomplished, a date is entered, giving management their Y2K project history. Augmented with lessons learned, these project histories will support future maintenance efforts and potential litigation that may arise due to unforeseen Y2K noncompliance.

Summary of Services

The STSC reengineering and Y2K staff provides on-site consulting services on a cost-recovery basis. These services address all phases to upgrade systems to Y2K compliance, including

Awareness

- Y2K management briefings and tutorials.
- Test, inspection, and Y2K workshops and seminars.
- Y2K vulnerability assessments.
- Automation support (evaluate, select, and adopt).
- Y2K strategic planning and project management.

Assessment

- Systems, interfaces, and documentation inventories.
- Y2K system project plans (develop and review).
- Risk analysis and contingency planning.
- Code Y2K impact analysis (manual and tool assisted).
- Initial date-function tests of vendor-supplied software.

Renovation

- Code correction (manual and tool assisted).
- Code inspections and unit testing.
- Code correction progress tracking.

Validation

- Test plans and test procedures (prepare and inspect).
- Test execution (manual and tool assisted).
- Test progress tracking.

Implementation

- Operational test, evaluation, and certification.
- Coordinating production system implementation.
- Update project documentation and collect historical data.

We also have a number of resources that the STSC can provide to organizations to support their Y2K efforts, including

- Top management and project-specific Y2K guidebooks.
- Preliminary analysis of Y2K planning documents or test plans.
- Y2K team augmentation (assistance) services.
- Project planning.
- Source code analysis.
- Source code renovation.
- System testing.
- Certification and compliance.
- Customized tool lists from one of the industry's largest Y2K tool databases.
- Guidance in tool evaluation, selection, and adoption.

Points of Contact

For more information, please contact the Y2K team.

Paul Harames
haramesp@software.hill.af.mil
801-775-5555 ext. 3091 or DSN 775-5555 ext. 3091

Greg Daich
daichg@software.hill.af.mil
801-775-5555 ext. 3043 or DSN 775-5555 ext. 3043

CROSSTALK

Sponsor	*Lt. Col. Joe Jarzombek* 801-777-2435 DSN 777-2435 jarzombj@software.hill.af.mil
Publisher	*Reuel S. Alder* 801-777-2550 DSN 777-2550 alderr@software.hill.af.mil
Managing Editor	*Tracy Stauder* 801-777-9239 DSN 777-9239 staudert@software.hill.af.mil
Senior Editor	*Sandi Gaskin* 801-777-9722 DSN 777-9722 gaskins@software.hill.af.mil
Graphics and Design	*Kent Hepworth* 801-775-5555 ext. 3027 hepwortk@software.hill.af.mil
Associate Editor	*Lorin J. May* 801-775-5555 ext. 3026 mayl@software.hill.af.mil
Editorial Assistant	*Bonnie May* 801-775-5555 ext. 3023 mayb@software.hill.af.mil
Features Coordinator	*Heather Winward* 801-775-5555 ext. 3028 winwardh@software.hill.af.mil
Customer Service	*Barbara McDonald* 801-777-8045 DSN 777-8045 mcdonalb@software.hill.af.mil
Fax	801-777-8069 DSN: 777-8069
STSC On-Line	http://www.stsc.hill.af.mil/
CROSSTALK On-Line	http://www.stsc.hill.af.mil/ Crosstalk/crostalk.html

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address:

Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205

E-mail:	custserv@software.hill.af.mil
Voice:	801-777-8045, DSN 777-8045
Fax:	801-777-8069, DSN 777-8069

Editorial Matters: Correspondence concerning *Letters to the Editor* or other editorial matters should be sent to the same address listed above to the attention of *CROSSTALK Editor* or send directly to the senior editor via the E-mail address also listed above.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the *Guidelines for CROSSTALK Authors*, available upon request. We do not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Most material in CROSSTALK may be reprinted at no charge. Coordinate reprint requests with CROSSTALK.

Trademarks and Endorsements: All product names referenced in this issue are trademarks of their companies. The mention of a product or business in CROSSTALK does not constitute an endorsement by the Software Technology Support Center (STSC), the Department of Defense, or any other government agency. The opinions expressed represent the viewpoints of the authors and are not necessarily those of the Department of Defense.

Coming Events: We often list conferences, seminars, symposiums, etc., that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC On-Line Services: STSC On-Line Services can be reached on the Internet. World Wide Web access is at http://www.stsc.hill.af.mil/. The STSC maintains a Gopher server at gopher://gopher.stsc.hill.af.mil/. Its ftp site may be reached at ftp://ftp.stsc.hill.af.mil/. The Lynx browser or gopher server can also be reached using telnet at bbs.stsc.hill.af.mil or by modem at 801-774-6509 or DSN 775-3602. Call 801-777-7989 or DSN 777-7989 for assistance, or E-mail to portr@software.hill.af.mil.

Publications Available: The STSC provides various publications at no charge to the defense software community. Fill out the Request for STSC Services card in the center of this issue and mail or fax it to us. If the card is missing, call Customer Service at the numbers shown above, and we will send you a form or take your request by phone. The STSC sometimes has extra paper copies of back issues of CROSSTALK free of charge. If you would like a copy of the printed edition of this or another issue of CROSSTALK, or would like to subscribe, please contact the customer service address listed above.

The ***Software Technology Support Center*** was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies that will improve the quality of their software products, their efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery. CROSSTALK is assembled, printed, and distributed by the Defense Printing Service, Hill AFB, UT 84056. CROSSTALK is distributed without charge to individuals actively involved in the defense software development process.

Year 2000: The MacGyver Solution

I've been talking to year 2000 (Y2K) experts and reading the recent Y2K analyses of heavyweights like Capers Jones, Ed Yourdon, and Peter de Jager. Although they present what looks like irrefutable evidence that much of the world will be unprepared for the Y2K crisis, unfortunately they ignore one critical factor that should cause us all to reject their gloomy forecast: nothing as bad as a worldwide computer failure could ever happen! Not in a million bajillion years! It can't! It can't! IT CAN'T!

In spite of this kind of strong data, many Y2K "experts" insist that most organizations have started far too late to fix all their systems. It is sad that just because these experts have watched a few hundred non-Y2K-compliant systems crash and burn, and have seen how long it takes just to test a system, and have seen that organizations are allocating only a fraction of the needed resources to the problem, they think they can do some math and declare that almost everyone is ridiculously behind schedule.

These experts obviously haven't watched enough television. By now, everyone should know that no problem is too big, and no odds are too high for a scrappy group of misfits who appear a little rough around the edges, yet when the going gets tough they can do the impossible. I can't count how many times I saw "The A-Team" take only one afternoon to create, for example, a tactical stealth helicopter using nothing but duct tape and an abandoned '76 Pacer. Certainly, *our* computer engineers could muster an equivalent feat! I bet that MacGyver alone could fix the Y2K problem using nothing but PVC pipe, a transistor radio, and Gouda cheese.

But I won't bore you with the technical details of Y2K fixes. For the sake of argument, let's pretend that, as the experts predict, by Jan. 1, 2000, a large portion of the world's computers will be spitting out *even more* garbage data than they do right now. Let us assume that non-Y2K-compliant microchips cause everything from microwaves and elevators to airplanes and bank vaults to start malfunctioning. As I will prove below, any nitwit can refute the worst-case projections of the experts.

Businesses will have a greatly decreased ability to produce, distribute, and sell goods. Rippling financial losses may trigger stock price plunges, business failures, bank closures, unemployment, and recession. Yeah, right. Do they really think the world economy is that volatile? Look at the U.S. stock market—stable, independent, impervious to any outside influence other than when Alan Greenspan sneezes in the wrong direction. And who cares if all that highly overvalued stock suddenly self-corrects to a fraction of its present value? The Great Depression wasn't *that* bad.

The communication, distribution, and utility infrastructures may limp badly for a time. (Yawn.) What if it did happen? Most people have a few days' supply of food on hand and plenty of flammable furniture. And we're talking about the dead of winter here—plenty of snow to melt into water. A few days is time enough for thousands of food distributors and utility companies to work out any unanticipated Y2K bugs from their decentralized, real-time networks run by thousands of interconnected, one-of-a-kind computer systems. Not to worry!

State, local, and federal government computer systems will be largely unable to function. This could result in many wholesale changes to the way taxes are collected and services are rendered. That's supposed to be a disadvantage? It goes without saying that in the face of disaster, our elected leaders would set aside petty differences, and in less time than it takes to vote down a congressional pay cut, would form a solution that is efficient, effective, fair, and beneficial to all.

I could go on and on, but I've made my point. And one more thing—if the experts are so sure this "crash" will cause such chaos, may I ask: Are *they* working frantically to safeguard their organizations' survivability? Are *they* storing food and fuel for a worst-case scenario? Are *they* avoiding debt and putting their wealth into tangible assets? The answer is yes, they are. But what do *they* know? It all comes down to whether you believe we can grit our teeth and somehow do the impossible (Rah! Rah!), or you think it's time to face the music and start working on feasible system contingency plans (Boo!). So who do we bet the farm on: MacGyver or the "experts"? I'll take MacGyver—after all, what have we got to lose?

—Lorin May

Got an idea for BACKTALK? Send an E-mail to mayl@software.hill.af.mil