Short-Term Fixes Shade Future

Forrest Brown
Managing Editor

I recently asked an elderly gentleman if he had heard of the year 2000 (Y2K) problem that will soon impact many computer systems. He said he had not. After I briefly explained the Y2K problem, he emphatically stated, "Ours is a generation of abbreviations, and we are always in a hurry."

I realized how right he was when I headed to work the next day. On the radio, the announcers were talking about baseball's AL and NL MVPs, whether the current lockout would KO the NBA, and the impact it would have on the NFL and NHL. I went in the office and powered up my PC's CPU so I could check my E-mail to see if an author had received my fax about an article that discussed the ROIs associated with COTS. As I booted, I glanced at a fed IT magazine that discussed the FY99 DoD authorization bill (HR-3616) and a legal battle between AT&T and GSA. I looked in my day planner and saw notes on getting X-mas cards and the date of our next CEB. In the tech world, we talk about SEPGs doing CBA IPIs and SCEs and pushing SPI throughout the whole org. My business card says I work at the STSC at Hill AFB, UT.

Is it any wonder that we software developers brought this problem on ourselves? We even refer to the year 2000 problem as the Y2K bug, just to save a few characters. We need not look down on computer programmers in the '60s and '70s (sorry: 1960s and 1970s) for trying to save space in systems that were hard pressed for memory allocation. Who knows what shortsighted software decisions we're making today (including our Y2K renovation choices) that we will regret tomorrow?

It is worth noting that more than mainframe computers will have problems next year. For example, the world's consensus standard operating system, Windows, is not compliant. In "Time To Debunk Y2K Myths" (*Information Week*, Sept. 28, 1998, p. 172), Leon A. Kappelman states, "The name alone should alert us to the simple fact that *Windows* 98 *is not* Y2K OK, *either.* Windows 98 defaults to two-digit years just like Windows 95, and two-digit years can lead to problems. In fact, all versions of Windows have date-processing problems. ... The same is true of practically every Microsoft product, including the newest versions."

Many questions remain as we approach January 2000. At this point, the question is not whether we can fix all the world's computers in time (we cannot) but whether we will finish renovating the most critical systems and fix them correctly. To make the right decisions, we need a clear idea of which systems need to be fixed and which can be left alone or left to die. Patricia McQuaid and Lee Fischman note in their article (page 11) that "One big mistake made in scoping Y2K renovation is assuming that all legacy software needs treatment."

As we hurry to renovate our systems, we cannot just work fast, but must work smart. When we get to January 2000, let us hope we have not made unwise renovation shortcuts, work-arounds, and other "abbreviations" that do not work, but put us deeper in the hole than we already are. The sacrifice of long-term needs for short-term savings is what put us in this mess in the first place. ◆

Call for Articles

If your experience or research has produced information that could be useful to others, *CROSSTALK* will get the word out. 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 1999 *CROSSTALK* issues. In future issues, we will place a special, yet nonexclusive, focus on

Metrics and Measures
June 1999
Article Submission Deadline: Feb. 1, 1999

Project Management, Cost Estimation, Risk Management
July 1999
Article Submission Deadline: March 1, 1999

Software Acquisition Management
August 1999
Article Submission Deadline: April 1, 1999

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; 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.

Ogden ALC/TISE
ATTN: Denise Sagel
CROSSTALK Features Coordinator
7278 Fourth Street
Hill AFB, UT 84056-5205

Or E-mail articles to features@stsc1.hill.af.mil. For more information, call 801-777-9239 DSN 777-9239.

COMMAND, CONTROL, COMMUNICATIONS, AND INTELLIGENCE

OFFICE OF THE ASSISTANT SECRETARY OF DEFENSE
6000 DEFENSE PENTAGON
WASHINGTON, DC 20301-6000

23 Sep 1998

MEMORANDUM FOR SECRETARIES OF THE MILITARY DEPARTMENTS
CHAIRMAN OF THE JOINT CHIEFS OF STAFF
UNDER SECRETARIES OF DEFENSE
DIRECTOR, DEFENSE RESEARCH AND ENGINEERING
ASSISTANT SECRETARIES OF DEFENSE
GENERAL COUNSEL OF THE DEPARTMENT OF DEFENSE
INSPECTOR GENERAL OF THE DEPARTMENT OF DEFENSE
DIRECTOR, OPERATIONAL TEST AND EVALUATION
ASSISTANTS TO THE SECRETARY OF DEFENSE
DIRECTOR, ADMINISTRATION AND MANAGEMENT
DIRECTORS OF THE DEFENSE AGENCIES
CHIEF, NATIONAL GUARD BUREAU

SUBJECT: Year 2000 (Y2K) Compliance—FY 1999 Reporting Requirements

The secretary's memorandum entitled "Year 2000 Compliance," dated August 7, 1998, directs several measures by specific dates to improve the accountability for Y2K compliance. To carry out the secretary's requirements, the OSD staff developed an implementation plan that underscores the need for accurate mission-critical information in the DoD Y2K database. Guidance for the Acquisition Category I, IA, II system reports will be provided separately, since this information will not be used in the determination of Y2K-related FY 1999 funding withholds.

The military departments, the commanders-in-chief, and the defense agencies are responsible for consistent, accurate, and timely submission of Y2K information for the DoD Y2K database. In this regard, the data reporting requirements found in the ASD (C3I) memorandum, "Year 2000 Assessments," dated August 1, 1996, and the ASD (C3I) memorandum, "System Interfaces, Data Exchanges, and Defense Integration Support Tools," dated November 5, 1996, no longer apply. While my staff is available to assist you in your actions to ensure the accuracy of the Y2K database, each component is responsible for all actions necessary to eliminate redundancies and inaccuracies of system and subsystem reporting. Each component must ensure adherence to my memorandum entitled "Year 2000 Database Reporting," dated June 19, 1998. The updated and corrected Y2K database as of October 1, 1998 will serve as the baseline for Y2K mission-critical systems.

To further comply with the secretary's direction, additional information is necessary to record and maintain the status of formal interface agreements for Y2K compliance. Each component is responsible for determining that there is a complete set of interface agreements for all pertinent mission-critical systems under its purview. Defense Information Systems Agency (DISA) will provide by September 25, 1998, to my Resource Management office a list of all megacenter domains and associated domain users who have failed to or are not planning to sign explicit test agreements with DISA by October 1, 1998. DISA will provide a copy of this list to each affected DoD component. DISA will also provide the estimated resources provided by these domain users.

As part of the department's upcoming apportionment process, the Office of the Under Secretary of Defense (Comptroller) will direct the services and defense agencies that all funds for each mission-critical system be withheld from obligation until the system meets the standards of the Secretary of Defense memorandum. On a case-by-case basis, I will consider granting waivers for individual systems that are necessary to ensure the performance of essential military functions, or that are based on safety considerations. Withheld funds will be released through the normal fund release process. My staff will hold period reviews on the status of Y2K withhold funds.

Prior to obligation of funds, the military departments, the commanders-in-chief, and the defense agencies are responsible for making sure that any contract that processes date-related information contains the Y2K requirements specified in Section 39.106 of the Federal Acquisition Regulation.

To assist you in these efforts, additional guidance documents are available on the C3I Web page (www.dtic.mil/c3i/). For comments and questions, my point of contact is Ms. Sally Brown, 703-602-0967, sally.brown@osd.pentagon.mil.

Arthur L. Money
Senior Civilian Official

Improving Software Engineering Practice

Patricia Sanders
Office of the Undersecretary of Defense for Acquisition and Technology

The complexity and size of the Department of Defense (DoD) necessitates an extensive, software-dependent computer network; however, past experience has shown that software is rarely defect free. In an organization that requires pinpoint accuracy to save lives, functional software is an area in which excellence cannot be compromised. The only *way to improve software's performance is to improve the way in which software is developed.*

Increasingly, we live in a complex world where software is indispensable to everyone's life. And increasingly, we are frustrated by the software with which we are forced to live.

Consider if a carpenter were compelled to work with tools as unreliable, complex, and generally inaccessible as most of our computers. Can you imagine turning on a sander and getting a message that says, "general protection fault," at which point the disk flies off and the whole thing self-destructs? Why do we put up with this from tools which, for many of us, are as indispensable as a sander is to someone who makes a living by woodworking?

If we want quality software, we must accept responsibility for how it is developed. Improving *what* you build means improving *how* you build.

The DoD's Environment

The DoD is a large and complex organization. There are 1.4 million active duty men and women in the uniformed services and about 800,000 civilians. Every year, we recruit around 200,000 new people to join the armed forces and separate about 220,000. So approximately 30 percent of our organization is either coming or going every year. We have about 250 major installations worldwide. We operate 550 public utility systems—gas, water, electricity, and natural gas distribution.

We support one of the larger school systems in the world comprising 126 high schools and elementary schools. We are the world's largest day-care provider—300,000 children are enrolled in DoD day-care centers.

This article is based on a speech given at the 1998 Software Engineering Symposium in Pittsburgh, Pa. Sept. 16, 1998.

We have 28,000 separate computer systems that we are tracking for the year 2000; 2,800 of them are mission critical. We disperse 5 million paychecks and about 400,000 bonds and about 600,000 travel vouchers and 800,000 contract actions every month. In Columbus, Ohio, where we do our large contract management administration, there are about 390,000 contracts under administration. We disburse the staggering amount of about $43 million an hour.

We sustain operations in every time zone. Today, there are about 120,000 military personnel deployed around the world, in addition to the 200,000 who are permanently stationed overseas. We operate over 400,000 vehicles—everything from sedans and buses to the street sweepers used to clean runways to combat vehicles to tanks to armored vehicles.

As an organization, one of our challenges is to concurrently manage about 70 years of technology. We operate, on a daily basis, aircraft that were designed back in the early 1950s, and we still have to maintain them, buy spare parts for them, and keep them updated. At the same time, we are working on research and development programs for systems that will not be fielded until between 2015 and 2020. To manage that spectrum of technology is a constant challenge.

In information technology, we operate some of the world's most advanced computers, and yet, just last year, we moved a number of Burroughs punch card readers to a new megacenter because we are still operating punch cards for some business applications. We manage an astounding spectrum of technology. The DoD is not only the largest but also is probably the most complex organization in the world.

Yet, this is an organization that has had its budget cut for 15 consecutive years, has undergone significant reductions, is operating at 46 percent of the budget resources it had only 12 and 13 years ago, and has a third of its personnel coming and going in any one year. And still, it is an organization that is able, within a month, to send 60,000 people to the Persian Gulf along with 400 combat aircraft and 500 cruise missiles and could carry out war tomorrow if necessary.

War-Fighting Changes

We have been engaged in an unprecedented change in the way we think about warfare. It has been going on for some time, and it is moving into a highly sophisticated dimension. The change accelerated in the late 1970s and the early 1980s when we were starting to bring microprocessors into weapons systems.

We are on the edge of breaking through in what we call *network-centric warfare*. To put it bluntly, the DoD is in the business of destroying things. In warfare, we try to do that in a focused way without doing a great deal of damage to things we do not want to destroy. We have done that before by putting extremely lethal and highly accurate capabilities in the hands of whoever was doing the shooting at the time. We are now moving into a more interesting and highly leveraged dimension where the person who launches the missile does not have to see the target. We are going to be sharing information across a network and still be able to attack and destroy an opponent. This dramatically improves the survivability of our own forces, of

course. It is going to be revolutionary. The situational awareness that will be on our side of the battlefield will be three or four orders of magnitude better than our opponent's. We call this *information dominance*.

In the past, the dilemma of warfare was always how to bring mass together for its effect over your opponent without giving your opponent lots of targets at which to shoot. It is the classic dilemma. One of the reasons there were so many casualties during the Civil War was that firepower technology had progressed so much farther than communications technology. We were still massing people close to each other, side by side, so the soldiers could hear shouted orders. Firepower technology had advanced, however, so that cannon could mow people down. We are now going to be in a wholly different world where people do not have to see each other and yet, they can operate together as a combined arms team. What we expect to be able to do is quite dramatic.

Importance of Software

So a new breed of "knowledge warriors" has begun to emerge who recognize that knowledge can win or prevent wars. And this is causing fundamental changes in what is important to our war-fighting capability.

Today, about 10 percent of the weight and one-third of the cost of modern combat aircraft are composed of electronics and related components. Principal among the latter is software—a substance that weighs nothing but costs inordinately.

There has been nothing like the headlong rush to software since the similar rush to electronics after World War I. The average automobile of today has more software in it than the first Apollo spacecraft to arrive at the moon 30 years ago.

In the Gulf War, television cameras, ravenous for dramatic visuals, focused on F-14 aircraft roaring off the decks of carriers, Apache helicopters swooping over the desert, M-1A1 Abrams tanks growling over the sands, and Tomahawk missiles singling out their targets. Pieces of hardware became overnight stars. But the real star was the invisible software that processed, analyzed, and distributed data, though no television watcher ever saw those who produced and maintained it—America's software soldiers.

Software is changing military balances in the world. Today, weapons systems are mounted on or delivered by what we call "platforms"—a missile, an airplane, a ship, or even a truck. What we are learning is that cheap, low-technology platforms that are operated by poor, small nations can now deliver high-technology, smart firepower if the weapons are equipped with smart software. Stupid bombs can have their "IQ" raised by the addition of retrofitted components dependent on software for their manufacture or operation.

In previous eras, military spies paid special attention to an adversary's machine tools because they were needed to make other tools needed to produce arms. Today, the "machine tool" that counts most is the software used to manufacture the software that manufactures software, because much of the processing of data into practical information and knowledge depends on it. The sophistication, flexibility, and security of the military software base is crucial.

Software Costs

The DoD does not track software spending independently of other expenses. But, Federal Sources, Inc., a Virginia-based marketing firm that tracks government spending, completed a survey around September 1997, which concluded that by 2002, the DoD will spend more than $20 billion annually on software used for weapons systems, information technologies, and command, control, communications, and intelligence systems (not including personnel, management, and non-tactical systems). The Federal Sources review estimates military aircraft require by far the largest software expenditures, roughly $5 billion in 1998. Ships sail in at a distant second with barely more than $1 billion. Ordinance and weapons, lumped together in one category, tie for last with vehicles at less than $1 billion.

A study by the Electronics Industries Association estimated in 1995 that the DoD would spend $42.5 billion on computer systems, of which $35.7 billion would be on software—about two-thirds of that on maintenance. These analyses are important in that they illustrate the increasing reliance on software for warfare in the information age. Some, in fact, predict a future in which military hardware procurement becomes secondary to software purchases.

Costs of Software Failure

Information or knowledge superiority may win wars, but that superiority is exceedingly fragile. In the past, when you had 5,000 tanks and your enemy had only 1,000, you may have had a ratio of 5-to-1 superiority. In information war, you can have a ratio of 100-to-1 superiority, but it can all turn on a fuse or a lie or on your ability to protect your advantage from those who want to steal it.

The key reason for this fragility is that knowledge, as a resource, differs from all the others. It is inexhaustible. It can be used by both sides simultaneously, and it is nonlinear, which means that small inputs can cause disproportionate consequences. A small bit of the right information can provide an immense and strategic or tactical advantage, whereas the denial of a small bit of information can have catastrophic effects.

Pentagon leaders have been increasingly stunned upon learning that some of our computer systems can be and have been tampered with by hackers and by military exercises that demonstrate how easy it is for hackers to cripple U.S. military and civilian computer networks.

But my issue is not so much one of information assurance—although that is decidedly a top priority for the DoD, one with which the Software Engineering Institute (SEI) is providing major assistance—rather, I want you to focus on the implication that you succeed or fail on the software. It does not matter how much speed, or how much stealth, or how much armor plating you have; you will not succeed if the software does not work.

The cost of software failures can be high. In the commercial world, a system error in American Airlines scheduling software that incorrectly showed flights full resulted in a $50 million loss. System downtime for American Express costs $167,000 per minute; for Charles Schwab, the penalty is $1 million per minute.

The DoD's damages can be more expensive. Under the START II treaty (Strategic Arms Reduction Talks), three-quarters of our nuclear deterrent is in our fleet ballistic missiles, the effectiveness of which is in the hands of their fire-control software.

So, I contend that software that does not work is self-inflicted information warfare. The policies, processes, and practices that guide the development and use of information technology in general and software in particular are a crucial component of our strategy.

Expectations

Unfortunately, our overall track record for producing quality software is underwhelming. There is a perception that the DoD has a perfect record on software development—we never get it right.

According to the results of a study on U.S. software development reported by the Standish Group in 1996,

- In 1995, only 16 percent of software projects were expected to finish on time and within budget.
- In larger companies, only 9 percent of the software projects will be completed on time and within budget.
- An estimated 53 percent of projects will cost nearly 190 percent of their original estimates.
- Projects completed by the largest American organizations have only 42 percent of the originally proposed features and functions.

These findings show slightly better performance than an earlier DoD study. In that analysis of DoD software development projects that were originally estimated to take between two and three years to complete, there was, on average, a 36-month schedule slip, and one-third of all software programs were canceled before completion.

Despite the real and potential benefits software holds for us, expectations of software performance differ in interesting ways from expectations for hardware performance.

A story going around has it that at a recent computer exposition, Bill Gates reportedly compared the computer industry to the automobile industry and stated, "If GM had kept up with technology like the computer industry has, we would all be driving $25 cars that got 1,000 miles per gallon." General Motors reportedly addressed this comment by releasing the statement, "Yes, but would you want your car to crash twice a day?"

A similar story has it that if software engineers made automobiles, your car would sometimes die on the freeway for no reason, and you would accept this, restart, and drive on. Occasionally, executing a maneuver would cause your car to stop and fail, and you would have to reinstall the engine. For some strange reason, unlike a carpenter's tools, you would accept this, too.

This sort of reliability might be adequate in a word processor, but it hardly seems acceptable in a weapons system or where safety is a major consideration. After all, a soldier without a weapon is at best a tourist and more likely, a target.

Systems Engineering Process

To get good software, we need to build it right. When we track successful software developments, almost invariably, the accomplishment can be linked to the existence of good systems engineering processes because it is the application of the disciplined systems engineering process that makes the difference in achieving the functionality we seek—in both hardware and software.

As Reuel Alder observed (*CROSSTALK*, September 1998), "Discipline is no fun—I consider day planners self-inflicted torture. My idea of a good day is to wake up with no plan and accomplish more than humanly thought possible. The work would be intuitively discovered as the day progressed. Creativity and spontaneity would be enhanced, and routine, repetitive activities would be minimized. Each day would be a fresh and exhilarating experience filled with learning, personal growth, and development. The variations would be unlimited, and the success would be phenomenal.

"But if you believe the last 40 years of development data, this dream is not achievable for most software projects. Yet, we are still largely living in a dream world where we think software can be built by pure 'artists' who arrive at river's edge with no plans, and through sheer talent can turn a pile of scrap iron into a decent bridge.

"However, I have learned from unfortunate personal experience that almost all significant human achievements require more than just talent and creativity. Decades of data prove it: Even the best software artists do better work when they start with a foundation of planning, preparation, and discipline."

Consider requirements management. A 100-company survey by Standish Group International found that 45 percent of a software application's features are never used, 19 percent rarely used, 16 percent sometimes used, 13 percent often used, and 7 percent always used. Yet, in spite of the fact that most of an application is seldom used, software gets bigger all the time.

I have a cartoon in my office that shows two individuals—presumably software engineers—and one of them says to the other as he is running out, "You start coding; I'll go find out what they want." Unfortunately, there is all too much truth in this picture. Because what is being developed is "only software"—and everyone knows software is easy to change—a disciplined requirements management process is all too frequently lacking. Without requirements analysis upfront, however, the results are unsatisfied needs, wasted effort, and rework.

Software may be easy to change—at least relative to bent metal—but it can still be costly in both time and dollars. It is estimated that rework is 40 percent of the cost of development. Metrics collected by Capers Jones indicate that the cost and schedule impact of defects in requirements are the most expensive of all defects, followed by defects in top-

level design (architecture), and finally by defects in code.

We also do not develop software with its lifecycle in mind. Much of the software that is operational today will still be in service several years from now. Over the service life of software-intensive aircraft and smart munitions, there is a need for continuous improvement, correction, and addition of new capability via software modification. Embedded software in weapons system platforms has evolved in operational and technical impact to the point where upgrades must be seen as major subsystems. The effectiveness and efficiency of the process for upgrading and otherwise modifying embedded software has a major impact on readiness. Each year, upgrades to the B-1, F-15, and F-16 aircraft programs cost nearly $200 million. When the planned expenditures for the B-2, F-22, and F-117 aircraft and the advanced weapons are added in, that figure doubles.

One definition I have seen for software upgrade is you take old bugs out to put new ones in. As I previously noted, approximately 66 percent of the DoD's software costs are associated with maintenance. Almost all of the systems engineering practices that have high leverage for lowering the cost of maintenance are practices that need to be implemented during development. These include

- Development practices that reduce the density of defects in the software delivered into operation.
- Effective software test.
- A strong configuration management program.
- Taking account early in the program of the engineering environment and processes that need to be in place for sustainment.

SEI's Contributions

SEI has successfully influenced commercial technology for the DoD's benefit.

SEI's function, as defined in the DoD Management Plan, is to develop and transfer important technology to the private sector so that the government can benefit from a broader base of expertise. Their work benefits both the DoD and industry by helping to define, analyze, and improve operational processes from the level of the individual engineer to practices that apply across the entire organization. They have achieved measurable success.

SEI's mission is to reduce the cost, schedule, and technical and performance risk associated with acquiring and building software. Simply put, SEI exists to help us build software "better, faster, and cheaper."

But it must be predictably better, faster, and cheaper—erratically better, faster, and cheaper is not helpful to achieve the DoD's goals for information superiority. Discipline in process and product management is essential.

For fiscal 1999, we have worked with SEI to define some focus areas for initiatives.

- Commercial-off-the-shelf-based systems.
- Survivable systems.
- Architecture trade-off analysis and product-line practices.
- Continuing process improvement.

Conclusion and Summary

In closing, I will tell you a story about the brass lamp that Secretary of Defense William S. Cohen found in his office when he first moved into the Pentagon. When he rubbed the lamp, a genie popped out and offered him one wish (in a downsizing environment, you no longer get three wishes). Cohen first pointed to the large map covering the wall and the numerous pins in the map that indicated trouble spots around the world and asked the genie to provide stability to all those locations. The genie, however, said that this was perhaps too much even for a genie to accomplish. So the secretary thought some more and asked instead that the genie provide a guarantee of error-free DoD software. The genie, upon hearing this wish, said, "Let's look at that map again."

I am more optimistic.

The information technology revolution is having a profound effect on all of us. But never lose sight of the fact that all this progress depends on one fundamental: No matter how technologically sophisticated we are, it is people who make knowledge and knowledge sharing possible.

Real process improvement is not easy, and anyone who believes otherwise has never tried it or has never helped make an improvement of lasting significance. Learning better techniques and technologies is only the beginning—there are many human aspects through which to work.

Process improvement pays big dividends for those with the discipline to do it right. With it, we can improve what we build because we will have improved how we build. ◆

About the Author

Patricia Sanders is the director of test, systems engineering, and evaluation for the DoD, where she is responsible for ensuring the effective integration of all engineering disciplines into the system acquisition process. These include design, production, manufacturing and quality, acquisition logistics, modeling and simulation, and software engineering, with emphasis on test and evaluation as the feedback loop. She is also responsible for oversight of the DoD's Major Range and Test Facility Base and the development of test resources such as instrumentation, targets, and other threat simulators. She has over 24 years experience in the DoD. She holds a doctorate in mathematics from Wayne State University and is a graduate of the Senior Executive Fellow Program, John F. Kennedy School of Government, Harvard University.

POC: Brenda Zettervall
E-mail: zettterbt@acq.osd.mil
Voice: 703-695-2300

"The Network Is Down ..."

Capt. Cathy Walter
Headquarters, Air Force Communications Agency

This articles discusses critical network components that must be assessed for year 2000 (Y2K) compliance, what kind of errors to expect, and how to determine if the devices are at risk of Y2K faulty logic.

War fighters may laugh at the title, but the Air Force cannot support flying operations without its data networks. Until recently, the focus of Y2K efforts has been the renovation of mainframe-based software. But the day of the dedicated circuit and mainframe system is quickly disappearing. What am I talking about? Many Air Force command, control, communications, computers, intelligence, and logistics systems are designed to run on the Internet via the base's client-server data network, for example, the Global Command and Control System, the Global Transportation Network, the Core Automated Maintenance System, and the Standard Base Supply System. And do not forget basic E-mail service—who could survive without it? The Air Force air and space operational mission has become dependant on the information exchange E-mail offers.

Network device hardware and component software functions derive dates from embedded real-time clock (RTC) chips, basic input/output system (BIOS) firm or flash read-only memory, or network time-server hosts. These items pass date information to hosts, clients, network devices, and selected software applications. This article discusses the critical network components that must be assessed for Y2K compliance, what kind of errors you might expect, and how to determine if the devices you operate and maintain are at risk of Y2K faulty logic. A candidate list of network components for Y2K consideration is provided in the sidebar below.

Network Protocols

The underlying construct for all networks is the network protocols. According to a March 1998 paper by the Internet Engineering Task Force, most of the current implementations of these protocols will not be impacted by Y2K logic errors. However, transport layer, e.g., TCP, network layer, e.g., IP, application layer, e.g., HTTP, and protocol con-

Candidate Components for Y2K Consideration

Network Types

- Multiplexers with date-dependent channel or bandwidth controls.
- Routed networks with date-dependent routing metrics.
- Multicast radio nets with date-dependent controls.
- Point-to-point radio nets with date-dependent controls.
- Satellite networks with a shared date and time-dependent transponder.
- Cable networks with pre-programmed scheduling.

Network Systems

- Operating System services (Windows NT, HP UX, and NetWare).
- Domain Name Service (DNS) implementation.
- Network Information Service (NIS) implementation.
- Novell Directory Service (NDS) implementation.
- Network Time Protocol (NTP) Service implementation.
- Managed systems with date or scheduling processes (UNIX cron tabs).
- Automated backup systems.
- Messaging systems.
- Network management subsystems.
- Operational test equipment with date log.
- Dedicated server (Web proxy and time).

Devices

- Simple Mail Transport Protocol (SMTP) gateways.
- Managed modem banks (for remote access).
- Switches (voice, data, and video).
- Routers.
- Bridges.
- Firewalls or guards.
- Intelligent hubs.
- Multiplexers.
- Wireless controllers and managers.
- Date dependent CSUs and DSUs.
- Management modules.
- Token ring NIC cards using RTC functions.
- Any device that maintains time with an RTC.

Application Software

- Network, system, or application management utilities.
- Office automation.
- E-mail applications.
- Job control and scheduling.
- Scheduling.
- Metering software.
- Anti-virus software.
- Databases.
- Client-server applications.
- Date and time-dependent controls.

structs have not been fully investigated. At least one significant potential problem has been identified that may affect Web operations. Version 1.1 of HTTP, as defined by RFC (Request for Comment) 2068, requires the transmission of dates in a four-digit format as defined by RFC 1123. More than one-fifth of the Web servers on the Internet use a noncompliant two-digit format that was defined in the outdated RFC 850. Implementation of RFC 2068-compliant code will partially alleviate this problem. Older implementations of network protocols, e.g., RFC 850, that use two-digit dates are not Y2K compliant. For complete details, see The Internet and the Millennium Problem (Year 2000) on the Internet Engineering Task Force Web site at http://www.ietf.org/ids.by.wg/2000.html.

Another network construct of vital interest is the Network Time Protocol (NTP). It is used by network time-server applications to synchronize the date and time on all networked devices—from routers to supercomputers. The date format used by NTP has always been a four-digit format, so it is Y2K compliant. Devices that feed or use the date processed by NTP must adhere to the same date storage and maintenance format to also be Y2K compliant. In the Air Force, the date and time reference for NTP is drawn from traceable date and time references, e.g., Global Positioning System, and is compatible with Version 3 of the Network Time Protocol (RFC 1305). It is the transfer of incompatible, garbled, or truncated date and time information from network devices such as routers, application servers, e.g., time servers, and database hosts (Figure 1) that is a potential killer for our networks.

Workstations, Servers, and Minicomputers

The most common source of potential Y2K impact for networked data systems is from servers and workstations. Although the operating systems of networked devices may take the date and time from a network time server on boot-up, some applications tap the BIOS chip for this date information. Some communication system interfaces do not rely on a BIOS capability but use quasi-analog interfaces to track real-time clock transitions. What could it mean if the clock reference system, the digital BIOS or RTC is noncompliant? You could suddenly find you are no longer authorized to use your office automation system, your network connection is severed, or that your mission-critical networked files and E-mail have been deleted. An application server could grind to a halt under a flood of log data filling up a shared storage device. If you are fortunate, your resourceful local area network administrator or Network Control Center wizard may be able to recover and re-enter the corrupted user data. But what if the backup utility also fails, or you cannot wait for data restoration?

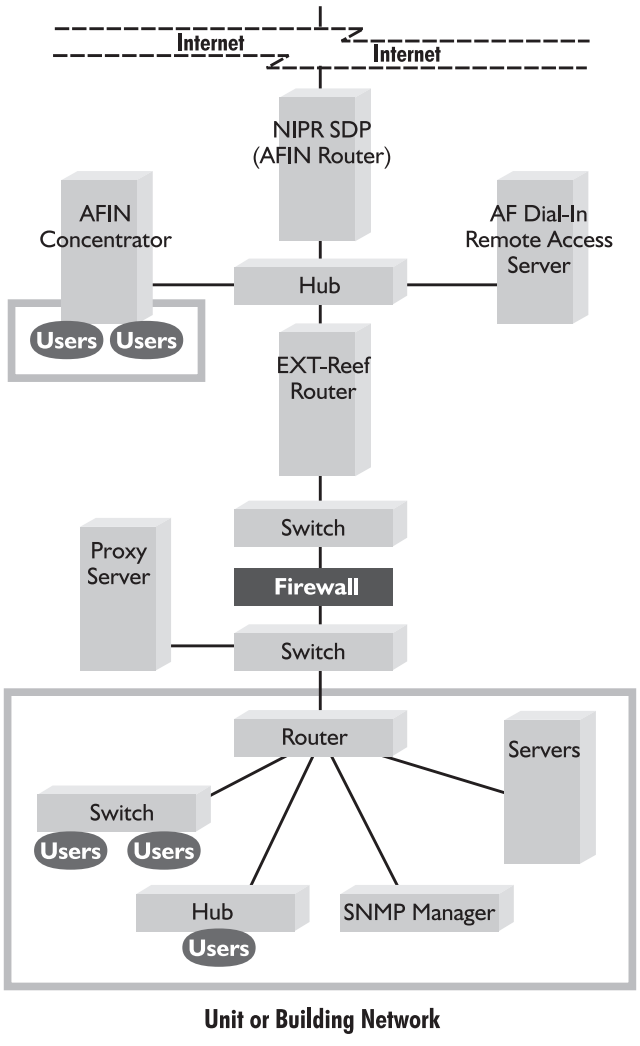
To give you an idea how pervasive and potentially critical this problem is, Figure 1 (items in gray) show where personal computers or workstations may be in use on your network. Note that noncompliance of any of these components could lead to serious problems during or after critical dates such as Sept. 9, 1999, Dec. 31, 1999, Jan. 1, 2000, and Feb. 29, 2000. One test conducted by the Air Force Communications Agency Technology Directorate found 35 percent of PCs will fail the Y2K rollover, but almost all of those can be fixed with a BIOS update or patch.

Fortunately, fixing a noncompliant PC is straightforward, since software patches and numerous utilities are available from original equipment manufacturers (OEM) and independent software vendors. However, before trying any new patch or utility, make sure you virus-check any new files or utilities you download, and back up all critical or time-sensitive data. Since some license agreements (especially on UNIX) may have expiration triggers, make sure you have validated access to all installed software before you attempt any date-change testing. If you have important but troublesome hardware for which none of the popular software patches work, RTC cards may be obtained for as little as $150 per machine. The RTC hardware replacement method is a viable option if BIOS updates or patches fail, or you do not want to rely on an external date and time clock reference.

Operating Systems and Applications

If the RTC or the BIOS chip does not get your network, the operating system or a faulty utility might. The most common

Figure 1. *The typical Air Force network topology.*



network operating systems in the Air Force, e.g., Windows NT and Novell NetWare, use different time utilities and synchronization methods to set the date and time on clients and servers. Beware that security functions, business applications, anti-virus software, management utilities, directory caching, and file and print services also rely on operating system dates. E-mail products, like other network-based services, are driven in large measure by date and time stamps. Messages may be deleted or rejected by the component utilities because the software thinks its shelf life has expired. Message measurement and tracking utilities also are vulnerable. Verify that your hardware is compliant and upgrade to a Y2K compliant version of the software.

Routers, Switches, and Hubs

On the diagram of a typical Air Force network (Figure 1), you will notice that if a network device is not a PC or workstation, it is probably a router, a switch, or a hub. A base may have anywhere from one to 50 of these devices, depending on the configuration of the base network. Routers, switches, and hubs do not require date and time stamps to forward packets, but operation and maintenance may experience problems if the associated operating software or controlling management application is noncompliant. Network events are logged in these devices, and the Y2K rollover could result in lost or misrouted error information.

Fortunately, most Air Force routers are the newer Cisco 4000 and 7000 models that can be upgraded into compliance through free patches available from the vendor. At some bases, older Cisco AGS/AGS+ routers also are prevalent. According to Cisco System representatives, these older routers are at the end of their lifecycle and will not be tested for Y2K compliance. Check your manufacturer's Web site for Y2K compliance and upgrade information. A listing of some key OEM Web sites are available on the Air Force Year 2000 Web site under Y2K Toolkit/Resources & Links.

Putting It All Together

After you have inventoried, determined compliance, and upgraded all these items, the next step is to perform operational assessments. Air Force functional communities are conducting functional integration and interoperability assessments to ensure mission continuity in the year 2000.

Focus on the Network

Air Force networks are mission critical to the operational mission of the Air Force. More and more deployed environments reach back to the fixed base architecture for communication support. It is also the one mission-critical piece of communications that has been wholly designed, funded, and acquired by base units. People at the bases are the only ones who know what is in use and can fix it. We have less than one year to ensure operations beyond the year 2000. Base Communication Squadrons must focus on this mission-critical asset.

How to Contact Us

For more about the Air Force Y2K program, contact us at the below numbers. You may also visit our Web page at http://year2000.af.mil for information on commercial-off-the-shelf compliance and testing and the status of centrally managed communication and information infrastructure items, e.g., voice switches. The site also includes Air Force Y2K guidance packages and links to other Web sites, including sites focused on the impact of Y2K on communications infrastructure. ◆

About the Author

Capt. Cathy Walter has been in the Air Force for nine years. She is the Air Force Y2K communication and information functional manager in the Air Force Y2K Program Management Office.

AF Y2K Program Management Office
HQ Air Force Communications Agency
203 W. Losey Street
Scott AFB, IL 62225-5222
Voice: 618-256-5761 DSN 576-5761
E-mail: AFCA-AFY2K@scott.af.mil

Inventory Your Network with Simple Network Management Protocol

Have you been trying to collect a complete list of network infrastructure components on your base for Y2K certification? Although this can be a difficult task, your network management system (NMS) can help get the project off the ground. HP OpenView, for example, is a common NMS with the capability to query network components for system information.

What information is needed to verify Y2K compliance for a device? The key information is node name, node type (router, hub, etc.), vendor, model number, and software version. An example description of a Cisco router would be "Bldg. 100, router, Cisco, 3000, IOS 11. 1(6)." NMS systems can be used to gather this data since it is contained in a standard management information base called system description, or sysdescr, on all simple network management protocol (SNMP)-manageable devices. You can use OpenView to check the system description by highlighting the component and selecting System Information from the Configuration menu.

A complete inventory requires information from nonmanaged network infrastructure components as well. The primary network managers at your base should be able to identify the key network equipment that is not visible on the NMS console. This can be traced to one of three causes: The device has no SNMP capability, SNMP has not been enabled on the device, or the community string for the device is not entered properly on the NMS. If the system cannot be managed using SNMP, the system data will have to be gained manually. Devices that are SNMP-manageable should be configured to be visible in your NMS not just for this project but to enhance overall network visibility and management. As a general security measure, ensure that community strings on your network equipment are not set to "public." See http://www.afca.scott.af.mil/pa/public/98may/intercom.htm to view other Air Force Y2K articles.

1st Lt. Mike Witzman
Staff Sgt. Kevin Nichols
Voice: 618-256-8513 DSN 576-8513
AFCA Scope Network

Estimating Y2K Rework Requirements

Lee Fischman, *Galorath Incorporated*
Patricia A. McQuaid, *California Polytechnic State University*

Especially at this late juncture, critical decisions regarding year 2000 (Y2K) projects need to be grounded in an understanding of the true scope of rework requirements for assessed systems. This article discusses how to characterize Y2K renovation work and what the scope of that work might be.

As we all know, the Y2K problem is so serious that it may affect you personally. If you are being caught late in the game, engaged in an emergency bout of planning, the first thing you should ask yourself is, "What is the scope of my problem?"

Galorath Incorporated estimates resource requirements for software development, particularly the staffing and the scheduling required to accomplish a particular software project. Recently, we have been asked to estimate a number of Y2K renovation projects because of our traditional expertise in modeling software modification and reengineering; Y2K work is in many ways an extension of such traditional work. However, we have experienced a learning curve of our own. Although you can and should apply the lessons of the past, Y2K work does have a language and concerns of its own.

In this article, we share our experience estimating Y2K work using the SEER-Year2K model we have developed. Every Y2K job is different, so you first need to carefully assess your software inventory before applying any model. When estimating a Y2K job, you also should not apply rules of thumb developed on the basis of macro, even national-level data, if you want an accurate assessment of your costs.

There are many different recipes to describe Y2K activities; our research approach was to merge commonly accepted Y2K renovation activities with those we have developed to categorize other forms of rework.

Legacy Sizing

To obtain good Y2K renovation estimates, you must measure the size of your legacy code. The two most common metrics for software size are function points and lines of code, both of which are supported in our method. However, for Y2K work, lines of code are usually the preferred measure: It is far easier to apply an automatic line counter to legacy code than it is to develop a laborious, manual function point count. Furthermore, changes in software will usually be made in a line-by-line fashion whether by automatic or manual methods.

It is more difficult or impossible to develop meaningful line-of-code estimates for items like database files; for these situations, we rely on more ideally suited function points. A size estimate may thus combine lines of code with function points when necessary. As long as there is no overlap in what is counted, this is perfectly legitimate.

If you are able to size a good amount of your legacy code, but not all of it, you can apply this knowledge to areas you know less about. This is called *estimating by analogy*. Simple analogies are as easy as saying, "This and that have a similar magnitude." There are more sophisticated analogy procedures that can produce risk ranges, in addition to producing more accurate estimates.

Effective Size vs. Legacy Size

One big mistake made when scoping Y2K renovation requirements is to assume that *all* legacy software needs treatment. There is, in fact, a difference between total legacy size and the effective size of the software undergoing reparation. Our model computes effective size through a series of *rework percentages*, which is discussed in the next section. Imagine effective size as the contents of a box, as depicted in Figure 1.

Although total size represents all the code you own, effective size is the code impacted by Y2K rework requirements. These requirements may involve simple testing or actual changes. Changes may be made manually or using a Y2K "solution" product. Much of the effort required for a good Y2K rework estimate is therefore involved in assessing the effective size.

When doing an overall estimate of rework required, there is a balance between details required and essentially macro knowledge. We have learned that this dichotomy can be embodied in overall size estimates that are then adjusted downward using sets of specialized percentages; we call this the *adjustment to effective size*, a process that is by no means straightforward. You need to first define the percentage items, then figure out how these percentages should be used to adjust the gross size estimate. We have acquired many data sets of projects completed, in addition to much heuristic knowledge and post-validations of our estimates, which have guided us toward proper definition of percentages and their formulaic specification.

Rework Items

This section describes rework categories for Y2K renovation that we have developed for our estimates. These originate from our knowledge of renovation issues, the experience of others, and our recent Y2K consulting engagements. Although the computations used are

Figure 1. *Effective size of the software undergoing remediation.*

specific to our model, the definitions should provide robust ground rules to evaluate your work. We have added detailed "Y2K Advice" sections to guide you in this. The rework items covered include

- Reverse Engineering Required.
- Date-Related Design Change.
- Specification Updates Required.
- Manual Recoding Required.
- Automated Recoding Required.
- Automated Conversion Verification.
- Programmer and Unit Testing Required.
- Test-Bed Preparation.
- Application Testing.

The diversity of these categories is a strength, since misspecification of any one category is not likely to drive the estimate too far off. The following sections provide details on these rework categories.

Reverse Engineering Required

This is the percentage of code (relative to the total application size) that a technical staff must review to understand what is happening at the code level. Include only those lines of code with which someone must be familiar for the application to be considered "reviewed."

Whole blocks of code may theoretically undergo "review," but lines thoroughly analyzed may be slight. In this case, the percentage of reverse engineering would only be the lines studied. Let us say that a "100 percent code review" translates into a scan across all code but only at the level of function calls—function contents are not examined. In this case, the percentage of reverse engineering is far lower than 100 percent—it may even be 1 percent or less.

To determine the percentage of reverse engineering, consider

- The level of familiarity with the system's internal logic. Systems that have been informally maintained over the years may now require basic understanding (ultimately expressed as flowcharts, entity-relationship diagrams, data flow diagrams, data dictionaries) before substantial renovation can begin.
- Formal documentation requirements may mandate additional reverse engineering, other than that necessary to complete work.
- Redocumenting requirements. These may be closely related to reverse engineering.

Y2K Advice

Outside teams may have to reverse engineer code to orient themselves to basic architecture and design. Code analyzers, scanning methods, and other automated tools may mitigate the need to reverse engineer—do not include the code covered by automated methods in this percentage. Older applications have higher levels of hidden utility and therefore require a more detailed approach, which often translates into additional reverse engineering.

Date-Related Design Change

These are date-related design changes measured relative to the total application size. Design is at a level that covers everything except actual programming.

To determine the percentage of redesign required, consider changes to

- Data structures, e.g., data-type changes, new or deleted fields, and expansions. When redesigning these, think in terms of the amount of code necessary to carry the design in a given data structure (such as in a structured query language CREATE command).
- Object methods.
- Date-related data passage between functions.
- Operating system-related issues, e.g., memory usage and date and time functions.
- Design changes at the function level (this does not include isolated code changes that do not change the function's design).

Y2K Advice

Consider the changes that actively address date issues. These include bridges to noncompliant external applications, encapsulation of potentially troublesome code to capture noncompliant dates, on-the-fly record format translation, windowing of two-digit years, redesign of date logic, representation, and manipulation.

Specification Updates Required

This is the percentage of new documentation to be developed compared to total existing documentation.

Redocumentation relates to both the proportion of a system being redocumented and the coverage of that documentation. An application may be well described at a high level, but this may only amount to a small percentage of what comprises the application. Only if requirements spell out that every single line of code be mentioned in new documentation is redocumentation 100 percent.

Redocumentation can also be thought of as the amount of code of which a technician must be cognizant to adequately document a system; in this way, it may be closely related to reverse engineering. Thus, a short report that describes a large system, which took only a few days to produce, is likely to fall into the low percentage ranges.

To determine the percentage of updates required, consider

- Comments. When inserted into the code, they are not part of formal documentation; however, this may be a part of reverse engineering.
- Documentation. If documentation must be completely rewritten, this does not necessarily mean 100 percent redocumentation. Documentation is calculated only with respect to the full application.
- Formal documentation requirements. Informal development shops with no formal processes or standards have documentation requirements that are typically orders of magnitude lower than those for shops that follow stringent standards.

Y2K Advice

Consider the state of existing formal documentation for this application, then decide whether it has to be updated. If there is no outstanding formal documentation requirement (have previous maintenance efforts had any?), the updates required percentage may be zero. An outsourcer may have formal documentation requirements imposed to ease later in-house maintenance or

because of other contractual requirements.

Manual Recoding Required

These are manual changes to software evaluated as a percentage of existing size. Thus, if 200 lines of a 10,000-line application are modified, recoding would be 2 percent.

To determine the percentage of recoding, consider

- New code. If any new code is being written to support changes, it should be factored into the size of the existing software to develop a correct percentage. Thus, if 200 lines of a 10,000-line application are modified and 200 new lines are written, recoding will be 4 percent.
- Language conversions. Major language changes, such as from COBOL to C (with no automatic conversion aids), will require 100 percent recoding, even if virtually no redesign is required.
- Minor changes due to a change in compilers.

Do not count code that is changed using an automated tool, but do consider the manual refinements that are necessary after the tool is used.

Y2K Advice

Consider code that directly impacts date and time issues. This includes, to the extent applicable, data typing and initialization of date and time variables, date logic, input and output of date and time data, encapsulation of existing code, bridges to noncompliant external applications, and other software patches necessary to ensure compliance, fault recovery, etc. A simple text scan helps reveal what percentage of the code needs attention and rewriting.

Automated Recoding Required

These are automated changes to software evaluated as a percentage of the existing size. Thus, if 500 lines of a 10,000-line application are to be modified by means of an automated tool, automated recoding would be 5 percent.

To determine the percentage of automated recoding, consider

- If any new code must be written to support the automated changes, it should be factored into the size of the existing software to develop a correct percentage. Thus, if 500 lines of a 10,000-line application are to be recoded by an automated tool, but 100 new lines must first be written to assist the automated recoding, automated recoding will be 6 percent, and manual recoding must be increased by 1 percent.
- If no automated tool is used, this input will be 0 percent.

Automated Conversion Verification

This is the amount of code processed by automated conversion that requires manual inspection, review, or walkthroughs. This should be given as a percentage of the code that is being converted by automated means. Code reviews are generally done to ensure coding standards and conventions are adhered to and to detect potential errors.

To determine the percentage of verification required, consider

- Code reviews are work that is subject to low-level, direct review by one or more people knowledgeable in the organization's standards and practices and the language in which the code is written.
- The use of automated code conversion may significantly reduce the need for code reviews or at least reduce the rigor of required reviews.
- If no formalized code reviews are performed, i.e., regular get-togethers, this could be 0 percent.
- Use of automated tools to check for adherence to coding standards and conventions may reduce or eliminate code reviews.

Y2K Advice

Are code reviews part of your normal development or maintenance process? If so, they will probably be part of your Y2K renovation. Code reviews are normally isolated to new changes. To develop an accurate percentage, consider the percentage of work that is normally reviewed and multiply this by the percentage of new coding.

Programmer and Unit Testing Required

This is measured as the percentage of code, relative to existing size, that requires unit testing. Unit testing is generally done by programmers to test and debug low-level software. Unit testing is usually considered at the module level (such as functions and subroutines) and the unit level (generally a source file).

To determine the percentage of unit testing, consider

- The amount of recoding required, because code changes typically must be tested. If 10 percent of code is changed and all of that is tested, unit testing is also 10 percent.
- External testing. If testing is carried out by people other than the programmer making code changes, these testers will probably cover more code than has been changed.

Y2K Advice

Are unit tests part of your normal development or maintenance work? If so, they will probably be part of your Y2K renovation. In addition, because unit tests of certain sensitive functions are an efficient way to track down faults, unit testing may exceed the amount of code changed.

Test-Bed Preparation

This parameter covers the preparation of new test plans and test procedures, not their implementation. Actual testing is covered in application testing. Test plans and procedures that already exist should not be included in the Test-Bed Preparation percentage.

Test preparation describes the scope, approach, resources, and schedule of test activities. It further identifies test items, features to be tested, tasks, who will perform each task, and any risks that require contingency planning. To clarify the difference between test procedures and plans, imagine an orchestra: Test plans describe the conductor's job, whereas test procedures describe the musician's instructions.

The percentage of test-bed preparation required relates to both the proportion of a system to be tested and the depth of testing required. In an

absolute sense, the overall percentage covers the extent to which the lowest logical attributes of the software are exercised. If plans spell out that every piece of code be reached by test plans and procedures, the percentage required is 100 percent.

To determine the percentage of test-bed preparation required, consider

- If you have informal integration testing and little or no formal testing, test-bed percentages are likely to be low.
- Test plans orchestrate testing activities but do not control the most detailed tasks. These are covered by test procedures.
- Can existing test plans be reused without modification? For every test plan that exists, somewhat fewer new plans may be required. The same may be true for existing test procedures.
- A clever test plan may simultaneously exercise multiple test points with a single directive. For instance, a repetitive software architecture may allow the test plan to specify an identical approach across system components. If each "test point" is separately accounted for in development of the test plan, coverage should include each test point separately. However, if only a single test point needs to be accounted for despite several being tested, coverage should include only that test point.

Y2K Advice

A Y2K renovation effort may require test plans and procedures if formal testing or third-party regression and integration testing is used. Furthermore, validation of date compliance may require that additional tests be drawn up.

Application Testing

This parameter covers only actual testing, not detailed test preparation. Preparation is covered in Test-Bed Preparation. Include all testing effort, even if familiar from previous efforts. Formal tests are conducted in an environment of intentional yet usually amicable mistrust; an outside authority or an in-house authority—which requires extremely formal turnover procedures—asks developers to provide proof that various aspects of a system work before they will accept delivery. Formal testing is sometimes called user acceptance testing.

The overall formal testing percentage covers the extent to which the lowest logical attributes of the software are exercised. Some formal tests provide an automatic environment that is designed to ultimately "contact" a high percentage of test points. If so, the percentage of formal tests required should be rated *lower* than the percentage of points that will be asymptotically "hit." The test percentage should instead be rated at the percentage of the application that the equivalent test effort could at *minimum* cover.

To determine the percentage of application testing required, consider:

- Do you do formal testing? If it is not a part of the standard product sign-off, formal testing will be zero.
- Complete formal testing does not necessarily imply "100 percent"—what percentage of code is actually exercised?

Y2K Advice

Meaningful formal acceptance testing is common in so-called "formal" development environments.

Rework Percentages

Recall that all the categories above are expressed in terms of percentages. The first "default" percentages that we developed were based on our experience with other types of renovation work; these can be modified by a user possessing more specific information. As a sanity check, however, it is useful to note that with these default percentages, our model yields results that are similar in magnitude to those produced by other third-party benchmarks, notably those by the Gartner Group and Capers Jones.

In Table 1, "Least" is the least likely coverage or percentage, "Likely" is most probable, and "Most" is the highest possible. These percentages are translated into effort and schedule estimates via SEER-Year2K's analytic model.

The percentages are in some ways quite general; what is important are

- The magnitudes being assumed.
- The balancing for the ranges specified.

It also is apparent that the percentages are quite low, emphasizing how strictly we have defined activities. This should make sense, because Y2K renovation falls far short of rewriting code. We

Table 1. *Rework percentages. Some are in hundredths because they are products of other estimating formulas.*

	Automatic			Semiautomatic			Manual		
	Least	Likely	Most	Least	Likely	Most	Least	Likely	Most
Reverse Engineering Required	0.00	1.00	3.00	0.00	2.00	6.00	0.00	4.00	12.00
Date-Related Design Change	0.00	4.00	4.00	0.00	4.00	4.00	0.00	4.00	4.00
Specification Updates Required	0.00	0.00	2.00	0.00	0.00	2.00	0.00	0.00	2.00
Manual Recoding Required	0.00	0.00	0.00	0.00	2.00	2.00	0.00	4.00	4.00
Automated Recoding Required	0.00	4.00	4.00	0.00	2.00	2.00	0.00	0.00	0.00
Automated Conversion Verification	1.00	5.00	10.00	1.00	5.00	10.00	0.00	0.00	0.00
Programmer and Unit Testing Required	0.00	1.14	1.30	0.00	2.58	2.65	0.00	4.00	4.00
Test-Bed Preparation	0.00	0.16	0.68	0.00	0.16	0.68	0.00	0.16	0.68
Application Testing	0.00	4.00	4.00	0.00	4.00	4.00	0.00	4.00	4.00

reiterate that these percentages are starting points; they need to be plugged into a formula to determine cost and—banish the thought with our millenium deadline—schedule. Over time and with enough projects, you could, as we have, develop such a formula—maybe by the millenium?

Conclusion

A diversity of activities are under the Y2K umbrella, and not all are always required. Sizing alone, therefore, cannot suffice as an accurate guide to effort required. It is useful to develop a set of secondary criteria based on types of activity.

You will need a standard way to specify the magnitude of Y2K rework activities; we have found that percentages are an efficient standard metric to scope activities. Percentages are well-suited to the gross inventorying that typically occurs in Y2K planning. Application specialists in the field with day-to-day responsibility for Y2K-impacted systems and the renovation team doing the work will buy in to the language of percentages with a minimum of introduction. Percentages thus give you an agreeable basis to rapidly approach an estimate.

The importance of a risk-based estimate cannot be overstated given the scarce resources and tight schedules of Y2K work. Without knowing upside or downside exposure, a simple point estimate in the face of such constraints carries tremendous risk. With a deadline that cannot be moved for these renovations, risk is not an option. ◆

About the Authors

Lee Fischman is special projects manager at Galorath Incorporated in El Segundo, Calif. He is active in the development of SEER tools and consulting methods. He wrote the Software Evaluation Guide for the Office of the Secretary of Defense, Program Analysis and Evaluation, and he has explored software economics and estimating in numerous papers over the past several years, all available at http://www.galorath.com.

Galorath Incorporated
Voice: 310-414-3222
E-mail: fischman@galorath.com
Internet: http://www.galorath.com

Patricia A. McQuaid is an assistant professor of management information systems at California Polytechnic State University at San Luis Obispo. She has taught a wide range of courses in both the business and the engineering colleges. She has industry experience in computer auditing and is a certified information systems auditor. Her research interests include software process improvement, software quality, and software testing, particularly in complexity metrics. She is the developer of a new software complexity metric known as the Profile Metric.

California Polytechnic State University
Management Information Systems Area
College of Business
San Luis Obispo, CA 93407
Voice: 805-756-5381
Fax: 805-756-1473
E-mail: pmcquaid@calpoly.edu

Coming Events

Configuration Management Seminars

Dates: Between Jan. 25, 1999 and March 5, 1999, depending on location and seminar.
Locations: San Diego, Calif., Las Vegas, Nev., Orlando, Fla., Washington, D.C.
Sponsor: Technology Training Corporation
Instructors: Robert Ventimiglia, Larry Bowen
Topics: Four seminars. Examples of topics: How to Integrate Configuration Management (CM) into Your Software Development Methods, the Latest CM Standards and Requirements, Establishing Appropriate Baselines.
Contact: Dana Marcus
Voice: 310-563-1223
E-mail: dmarcus@ttcus.com

ITCC World Conference and Exposition

Dates: Feb. 11-12, 1999
Location: Chicago, Ill.
Topic: Information Technology (IT) Consultants and Contractors (ITCC) World Conference and Exposition is the only industry event designed specifically for IT consultants and contractors. Join over 60 of the region's top IT consulting and software companies at the ITCC Exposition, along with a two-day conference program and workshops to maximize your professional success.
Internet: http://www.itccexpo.com

ESC Spring: Embedded Systems Conference

Dates: March 1-4, 1999
Place: McCormick Place South, Chicago, Ill.
Topic: Five days of high-level technical training as well as a three-day product exhibition. Target audience: embedded systems software developers, hardware engineers, and project leaders.
Internet: http://www.embedded.com/escfrm.htm

SEPG 99: 11th Software Engineering Process Group Conference

Dates: March 8-11, 1999
Location: Atlanta, Ga.
Subject: This four-day event brings together international representatives from government, industry, and academia for a global perspective on software process improvement.
Sponsor: Software Engineering Institute
Voice: 412-268-3007
Fax: 412-268-5758
E-mail: sepg@sei.cmu.edu

Effective Methods for Testing Year 2000 Compliance

William E. Perry
Quality Assurance Institute

Effective year 2000 (Y2K) testing must be performed using a process. The process for Y2K testing described in this article is based on experiences from many of the approximately 1,000 corporate members of the Quality Assurance Institute (QAI). By using this process, you will benefit from the experiences of leading corporations in addressing the Y2K problem. The nine steps in the process are designed to lead testers from the initiation of the Y2K testing effort to the writing of the final Y2K test reports and verifying the correctness of installing the Y2K changes into production.

Before developing a test strategy, tactics, and plan to test Y2K compliance, testers need to consider certain "concerns"—conditions that if present increase the probability that testing will not be effective. Identifying these concerns early and addressing them in the testing plan will help reduce the negative probabilities associated with those concerns.

The following 15 testing concerns are areas the testers need to address.

- Organization's track record on completing projects on time. If your information services organization has a history of missing scheduled dates, *there is a high probability it will miss the date for making Y2K changes.*
- Organization's track record on completing projects within budget. If your organization has a history of being over budget on software development projects, *there is a high probability it will not be able to complete Y2K changes with existing resources.*
- Maturity level of organization's development process at the time programs were written. If your organization developed systems that were not Y2K compliant with a Capability Maturity Model (CMM) Level 1 development process, *it will be much more difficult to change and test than projects written at Levels 2 through 5.*
- Currency of program documentation. If the documentation represents the code in the program, both changes and testing *will be easier than if the documentation is out of date and cannot be relied on by either the Y2K team or the Y2K testers.*
- Amount of program documentation. If the documentation for programs meets the documentation standards of standard-setting organizations such as the Institute of Electrical and Electronics Engineers, International Organization for Standardization, and the National Institute of Standards and Technology, *the ability to find and correct problems as well as conduct test methods will be significantly enhanced than in programs that are sparsely documented.*
- Amount of renovation to be included in the Y2K projects. If, in addition to making date changes, significant renovations, i.e., modifications and enhancements, are made, *they will increase the difficulty and potential problems associated with making and testing the changes.*
- Effectiveness of estimating procedures. If the organizations that estimate procedures are realistic, the estimates for both implementation and testing will be representative of the effort required; if the estimating process is not realistic, *there is a high probability that inadequate time will be available for testing.*
- Lack of skilled testers. The testers need the skills associated with testing the systems that are changed. They may require knowledge of obsolete languages, tools, and testing procedures. If the testers do not possess the necessary skills, *they will not be able to properly carry out the procedures.*
- Down-sized or burned-out staff. If information services has been understaffed for a time, *the staff may not be in a position to expend the extra effort needed to effectively complete Y2K changes on time.*
- Likelihood of litigation. If there is a high probability that inadequate Y2K compliance will result in litigation from customers, suppliers, stock holders, etc., *the testers will need to spend extra time and effort to both document the test processes and the results of the testing.*
- Lack of Y2K change tools. The individuals responsible for changing the Y2K date procedures need tools to help them search for the date problems and to make corrections. The fewer tools they are given, *the higher the probability they will not correctly identify and implement the needed changes.*
- Lack of Y2K test tools. Testers need the tools that will enable them to conduct one of the largest test assignments many will experience in their testing career. The lack of effective tools to test Y2K compliance *will inhibit the effectiveness of the testers and most likely increase the amount of resources needed for testing.*
- Importance of new projects. Experience has shown that many of the resources within an information services group will need to be diverted to make and test Y2K changes. If new projects have a higher priority for resources than the Y2K compliance project, *changes will be more difficult.*
- Lack of adequate testing resources. Estimates for testing Y2K compliance range from 50 percent to 75 percent of the total change effort. Finding those resources to perform the testing may be difficult. Lack of adequate testing resources *will result in shortcuts or omission of some changes in testing.*
- Lack of adequate testing time. One of the challenges testers face in any

software development project is that they are the last to work on the project. If the previous phases for Y2K compliance take longer than expected, *they will erode the amount of testing time available and thus reduce test effectiveness.*

Any Y2K test process that is going to be effective must address these 15 concerns. The concerns can be addressed by either resolving them immediately or incorporating appropriate tactics in the test plan to resolve or minimize the potential impact of the concern.

The Nine-Step Y2K Testing Process

The Y2K testing process used by the QAI follows the traditional "V" concept of testing (see Figure 1). The V shows the three major components of the Y2K correction process and the nine steps of the Y2K testing process. The first two steps of the correction process involve a verification process and test. The last step of the correction process is validation by an operational execution of the corrected software. The final step is the preparation of the report describing the results of implementation and testing.

A brief description of the nine-step Y2K testing process follows.

Step 1: Verify Y2K Assessment

The Y2K assessment scopes the size of the Y2K computing crisis and is a prerequisite to determining the effort required to correct the problem. Neither the implementation effort nor the testing effort can be determined until the scope of the problem is defined. It is the equivalent of the requirements or need definition component of new software development. During this step, the testers will challenge the completeness and correctness of the assessment performed to determine the scope of the Y2K computing crisis.

Step 2: Develop Y2K Test Plan

The scope of the Y2K test effort necessitates the development of the test plan. To expend the amount of resources needed for Y2K testing without a plan will probably lead to wasting valuable testing resources and the inability to make an evaluation of the status of the correction effort prior to Jan. 1, 2000. The test planning effort for the Y2K test project should follow the normal test-planning process; however, while the structure of the plan will be the same, the content will vary because it will involve not only software developed in-house but also supplier-developed software and software embedded into computer chips.

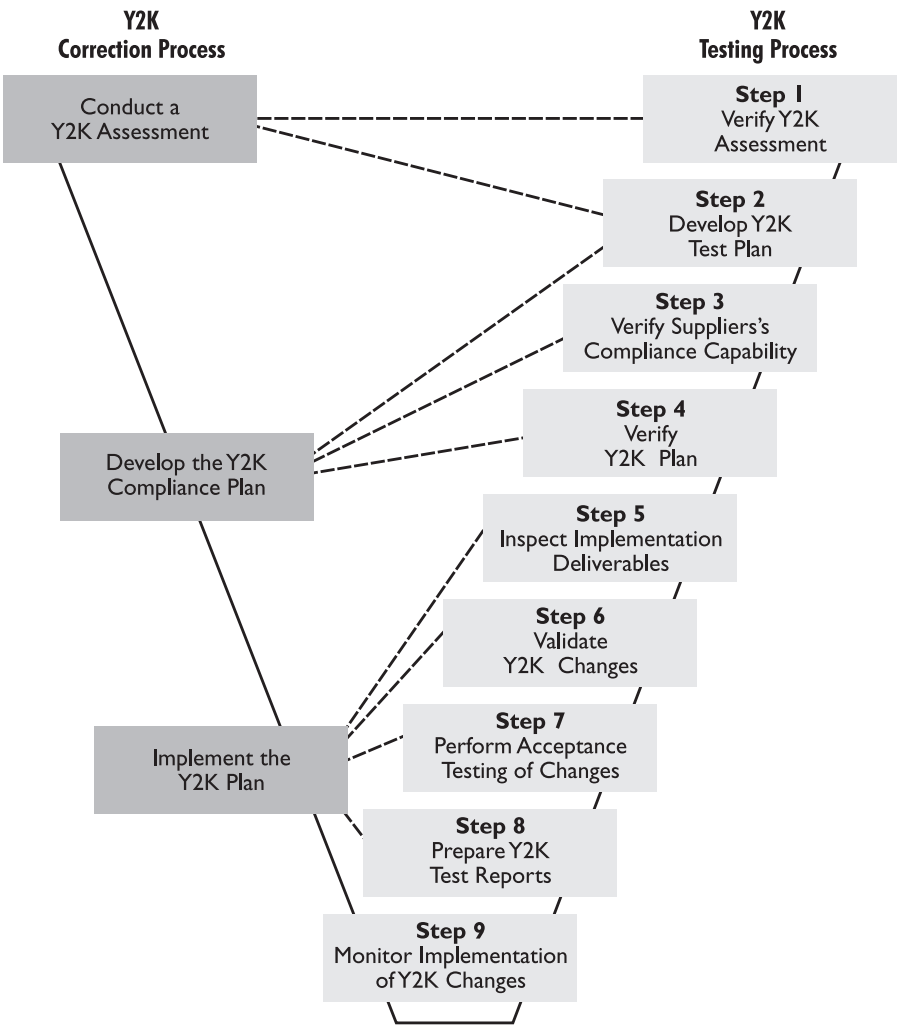
Step 3: Verify Supplier's Compliance Capability

Software provided by a supplier through purchase or contract poses the same Y2K computing crisis as software developed in-house. However, unlike software developed internally, organizations do not have direct control over supplier plans, projects, and employees. If the supplier has made Y2K corrections or built the software so that it is Y2K compliant, validation can be performed to determine whether that software is Y2K compliant. On the other hand, if the supplier is undertaking efforts to make software Y2K compliant, the execution of that software may not be known until almost Jan. 1, 2000. In the interim, at least for critical software, organizations should perform an assessment of the supplier's capability to make the software Y2K compliant.

Step 4: Verify Internal Compliance Capability

The organization's ability to achieve Y2K compliance on internally developed systems should be defined by the Y2K compliance plan. The more detailed the plan, the easier it will be to verify the adequacy and the completeness of the plan. In this step, a detailed review process to examine the plan will need to be implemented. The review process in-

Figure 1. *The nine-step Y2K test process.*



volves establishing a review team for its total competency and assessing all aspects of the plan. The review process will conclude with the preparation of a report, which will detail the strengths and weaknesses of the plan together with recommended improvements.

Step 5: Inspect Implementation Deliverables

In this step, the corrected software will be inspected prior to executing the software. The inspection process is used because first, it is more effective in identifying defects than validation methods; and second, it is much more economical to remove the defects through inspection than through unit or system testing.

Step 6: Perform System Testing of Changes

For this step, many types of test data that may be needed to perform effective Y2K testing will be identified. The test plan is decomposed into specific test transactions that will be used to validate the performance of the operational system. It is assumed that the developers will perform unit testing.

Step 7: Perform Acceptance Testing of Changes

The system testing will evaluate the functional and structural components of the software that has been changed. It will attempt to determine that the systems are Y2K compliant. Acceptance testing is a test performed from a user perspective. Acceptance testing may be necessary for any of the following three reasons.

- Potential Regression. The system may not perform the tasks that were performed previously because in correcting the Y2K date problem, other processing components could be negatively impacted.
- Enhancements. Many organizations will make enhancements to the system at the same time they make the Y2K correction. This will not necessitate determining that the enhancements were corrected from the user perspective.
- Performance Changes. This system may process correctly but due to the Y2K date, change in performance may be negatively impacted. For example, in the correction process, it may increase the amount of time required to provide on-line responses, which could have a negative impact on the user's business.

Step 8: Prepare Y2K Test Reports

There should be both interim and final test reports. Interim test reports are needed for both testers and management. The testers need to know the status of testing and the status of defect identification and correction; management needs to know the status of the overall project effort. Management will also need to know, shortly prior to Jan. 1, 2000, the status of the Y2K change efforts and what risks the organization faces because of that status. This step is to enable the tester to verify that what was tested, reported on, and accepted has been correctly installed in a production status.

Step 9: Monitor Y2K Implementation Changes

In this step, the organization needs to address actions that testers could take that relate to the magnitude of changes that will be occurring during the Y2K correction process. For example, while date changes are being made, business may be needed for the software; therefore, one version of the software will be under development for date changes, while another version will be modified for business changes placed in the operation. In addition, changes to the process of making date changes will be incorporated. During this step, issues such as version control, change control, and control over the testing process during the dynamic change environment will be dealt with.

Summary

The Y2K program is expected to be the largest and most complex system conversion effort undertaken for many organizations. Because of the complexities and scope of the Y2K problem, it is critical that organizations develop comprehensive plans that establish schedules for all tasks and phases of the Y2K program, set reporting requirements, assign conversion or replacement projects to Y2K project teams, provide measures to assess performance, and anticipate the need for risk assessments and contingency plans.

Ironically, perhaps, the enormous challenge involved in achieving Y2K compliance is not technical; it is managerial. Whether organizations succeed or fail will be largely influenced by the quality of executive leadership and program management. Executive leadership sets the tone; program management makes change happen. It will be imperative for top management—including the chief information officer—to not only be fully aware of the importance of this undertaking but also to communicate this awareness and urgency to all employees in such a way that everyone understands why Y2K compliance is tremendously important. That urgency must also include planning and executing the test segment of the Y2K program. ◆

About the Author

William E. Perry has been the executive director of QAI since 1980. He is the author of more than 50 books, including *Surviving the Top Ten Challenges of Software Testing*, *Effective Methods for Software Testing*, *A Structured Approach to Systems Testing*, and *Year 2000 Software Testing*. A certified quality auditor and certified software test engineer (CSTE), he realized the need for a practitioner-based conference for software testers 18 years ago. This conference has been a favorite among testers ever since. He started the first certification for software testers, CSTE, at QAI. He has a master of business administration degree from Rochester Institute of Technology and a master of education degree from the University of Rochester.

Quality Assurance Institute
9222 Bay Point Drive
Orlando, FL 32819-7273
Voice: 407-363-1111
Fax: 407-363-1112
Internet: http://www.qaiusa.com

Y2K Web Sites

U.S. Air Force Materiel Command Year 2000 Bomb Squad Unit: http://www.cisf.af.mil/seit/seit/ProgMgmt/Y2kBSU/Y2kBSU.htm

U.S. Air Force Year 2000 Web Site: http://year2000.af.mil (military and government Internet domains only)

Software Technology Support Center: http://www.stsc.hill.af.mil/RENG/index.html#2000

Defense Information Systems Agency (DISA): Office of Chief Information Officer: https://www.dsdc.dla.mil/priv/projects/year2k/frontpg/y2khome5.htm

The Year 2000 Information Center™: http://www.year2000.com/cgi-bin/y2k/year2000.cgi

Chief Information Officers' Council Committee on Year 2000 Information: http://www.itpolicy.gsa.gov/mks/yr2000/cioy2k.htm

Federal Aviation Administration Year 2000: http://www.faay2k.com

Defense Information Infrastructure Common Operating Environment Y2K Compliance: http://coeeng.ncr.disa.mil/REFERENCE_PAGES/Y2K/y2k_table.htm

Congressman Steven Horn: http://www.house.gov/horn

The Year 2000 Support Centre: http://www.support2000.com

The Institution of Electrical Engineers: http://www.iee.org.uk/2000risk/updates/a01-5-4.htm

Year 2000: http://www.year2000.com

AntiY2K Associates Solutions: http://www.antiy2k.com

Electric Utilities: http://www.euy2k.com

Joint Position Statement on Embedded Systems and International Infrastructure: http://www.effectivebydesign.com/nutmeg/embedsys.html

Electric Power Research Institute: http://year2000.epriweb.com

National Association of State Information Resource Executives: http://www.nasire.org/ss/ST2000.html

Gary North: http://www.garynorth.com/y2k

The MITRE Corporation/Electronic Systems Center Year 2000 Home Page: http://www.mitre.org/research/y2k

Society for Information Management: http://www.year2000.unt.edu

Westergaard Information Center: http://www.y2ktimebomb.com

Computer Information Centre: http://www.compinfo.co.uk

Infinium Corporation: http://www.infinium.com/html/year_2000.html

Shakespeare and Tao Consulting: http://www.tmn.com/~frautsch/y2k2.html

CMP Net: http://www.techweb.com/wire/technews/year2000.html

"The Year 2000 Computer Crisis!": http://www.karinya.com/yr2k.htm

Microsoft Inc.: http://www.microsoft.com/technet/topics/year2k/default.htm

Federal Computer Week: http://www.fcw.com/ref/hottopics/y2k.htm

Year 2000 Managers' Toolbox: http://www.govexec.com/tech/year2000

Building Self-Reconfiguring Distributed Simulations Using Compensating Reconfiguration

Lt. Col. Don Welch, *U.S. Military Academy*
James Purtilo, *University of Maryland*

In distributed training simulations, simulators can lose their connections to the rest of the simulation. When this happens, the uncontrolled virtual entities exhibit unrealistic behavior. To avoid unrealistic behavior, the distributed simulation must reconfigure itself based on the state of the simulation software and the virtual world. We call the automatic restructuring of a distributed application with respect to a set of rules "compensating reconfiguration," and we have developed a software engineering environment that could be used to support its inclusion in Department of Defense (DoD) distributed simulations.

The DoD distributed simulation domain encompasses a variety of uses, architectures, and techniques. DoD uses distributed simulation for test and evaluation, analysis, and training. Each of these categories brings with it different requirements for the distributed simulation architecture. Currently, DoD has simulations that use a totally distributed approach, as discussed in [1] and [2] but has mandated that all simulations use a middleware approach as defined by the high-level architecture (HLA) discussed in [3,4,5]. HLA is designed to support a family of simulations such as uses mentioned above and aggregate, disaggregate, and component levels of detail.

Failures in distributed training simulations can cause unrealistic behavior. Should a simulator crash or lose its link to the rest of the simulation, virtual objects the simulator owns will continue under the control of their *dead-reckoning* algorithms until they are removed from the simulation. There are ways to provide more realistic behavior. Starting a new copy of the simulation on a different host can re-establish sanity if the simulator requires little or no human involvement. For human-in-the-loop trainers, this approach is not practical because it is too expensive to maintain simulators with crews that do nothing but wait around for failures. To substitute a computer-controlled simulator for the absent trainer is more cost-effective and also can successfully maintain simulation realism.

In some cases, a manned simulator would only be lost temporarily. When the human-in-the-loop system returns, it cannot simply be left out of the exercise as would be a computer-controlled simulator. The crew represents a significant investment in resources and training opportunity. To give the manned simulator control of its original objects will not always be appropriate, such as returning control of the original helicopter to a user when it has already crashed on the virtual battlefield. To reintroduce a simulator back into a simulation is a complicated decision that requires knowledge of the virtual world as well as the simulation configuration.

We call the automatic restructuring of a distributed application in accordance with a set of rules "compensating reconfiguration." We have developed a software engineering environment that could support its inclusion in DoD distributed simulations. The compensating reconfiguration component created through this environment imposes an extremely small performance penalty on the simulation and is not an unreasonably complex burden for the simulation builders.

Related Work

In the DoD distributed simulation domain, there has been an abundance of work that defines the HLA [3,4,5]. The HLA addresses the late joining, early departure, and changing ownership of federates (simulator components). However, fault tolerance does not seem to have been adequately addressed, and certainly it has not been addressed within the context of demands such as fewer support staff and human-in-the-loop simulations [1].

The gluing together of disparate heterogeneous distributed systems forms the foundation of HLA. Understanding interconnection abstractions like Common Object Request Broker Architecture [6] and Polylith [7] is critical to understanding HLA. Using standard interconnection abstractions makes the development of a software engineering environment practical. These abstractions make it possible for our framework to work with existing systems without resorting to changing any of the components. We feel that compensating reconfiguration is best built into the interconnection abstraction and provided as a service.

The end result of compensating reconfiguration is the dynamic reconfiguration of the application. There are two primary approaches to dynamic reconfiguration. The Conic approach moves the application to a quiescent state prior to reconfiguration [8]. This approach requires logic located in each component that will migrate a component to a quiescent state in finite time. This technique is more appropriate for simulations that do not run on wall-clock time. A virtual simulation cannot achieve a quiescent state and still maintain realistic behavior. C. Hofmeister's

approach is better suited for virtual simulations [9]. She requires that the components involved divulge their internal state, then loads this into the new component. Since simulators in a distributed simulation continuously divulge their internal state (which is most important to the rest of the simulation), the software is ready for dynamic reconfiguration without change.

N. Minsky has used laws to ensure consistency in the software architecture as it evolves. A set of invariant laws is enforced throughout the lifecycle of the software using independent monitoring [10]. We focus on keeping the behavior of the distributed system consistent throughout its execution.

Compensating Reconfiguration

Distributed simulations require dynamic reconfiguration to keep correct execution in the presence of external failures. The proper compensation for a failure is not always readily apparent. Making the correct compensation requires taking the current software and hardware configuration and status into account and can require the virtual world state and a mapping between the two. Current dynamic reconfiguration techniques provide only for considering system configuration and not the application state.

To compensate for an external condition can involve complex decisions. Straightforward, like-for-like substitutions are not always appropriate. To compensate reconfigurations involves heterogeneous changes to the simulation. By heterogeneous, we mean that a different type of simulator is substituted for the original. In the motivating example, it is impractical to keep a crewed simulator as the backup to the attack helicopter flight simulator. Another factor that adds to the complexity is that compensation decisions cannot rely solely on the current configuration of the distributed simulation. As shown in the motivating example, the internal state of the simulators must sometimes be taken into account. Since the system configuration and the simulator state are not static, the compensation logic must be dynamic, too.

The main concepts of compensating reconfiguration are first, mapping the virtual world state and system configuration; and second, using an abstract interface to build the decision logic. To maintain this mapping in software is complex. When requirements change, the more concentrated the code changes, the easier code changes are to make and the less likely they are to be in error. Using an abstract interface for the reconfiguration and compensation decisions allows the user to keep in mind the big picture and not become distracted by the dynamic reconfiguration implementation details.

We have built Bullpen, a tool to build compensating reconfiguration software in the distributed simulation domain. We named it Bullpen because as baseball managers must change their pitchers to meet the changing conditions of a game, our software must substitute simulators. Bullpen currently runs as an invisible support utility. It could just as easily be integrated into the run-time infrastructure if one is used. When it detects a condition of interest, Bullpen makes two decisions. The first decision determines the appropriate compensation for the condition. The second is how to dynamically change the structure of the distributed simulation to meet the desired configuration and maintain realistic simulation behavior.

Results

Our goal has been to produce compensating reconfiguration code with less effort that also is more accurate than using only a high-level programming language. In addition, we want to ensure that the compensating reconfiguration code can perform all the reconfigurations required by current applications. Finally, we want the execution speed penalty to be low enough to ensure that this is a practical approach.

We did a pilot study to determine whether our tool warrants further evaluation. In this pilot study, we used a number of different scenarios, all based either on military uses or military simulation exercises. For each scenario, we built the initial versions of the compensating reconfiguration software either in Java™ or with Bullpen. We then changed the requirements for the simulation and modified the software to match the new requirements. As we built and tested, we collected the metrics discussed below.

A lack of expressiveness in Bullpen's abstract interface would manifest itself in the worst case by our inability to perform one or more of the requirements changes. Since we were able to do all the changes from the scenario, Bullpen satisfies this provision. A less drastic lack of expressiveness would show itself through increased effort and complexity of the code needed to implement the changes. We did not face this situation, so we concluded that Bullpen is expressive enough as it stands.

We looked at six categories of requirements change. Changes to the reconfiguration interface—or the ways the reconfiguration code must interact with the distributed simulation infrastructure—are part of this category. An example is a new release of the run-time infrastructure with an updated application programming interface. Changes to the Reconfiguration Policy represent revisions to the choice of possible compensating dynamic reconfigurations. The Virtual Configuration category contains changes to the simulated world. This includes both the number and the associations between virtual entities. Likewise, changes to the System Configuration category include changes to the hosts or the software components that compose the distributed simulation. Changes to Virtual and System Configuration include requirements changes that involve the virtual world, the system configuration, and the mapping between them. The final category includes changes to the Conditions handled. An example of a condition is the return of a simulator. The Reconfiguration Policy and System Configuration are the changes the simulation builders are most likely to make during the prototyping phase. Changes to the Virtual Configuration are most likely to come from the users as they refine their concept for the simulation.

Change Category	SLOC	Time
Virtual Configuration (User Driven)	23%	20%
System Configuration (Prototyping)	35%	85%
Virtual Configuration and System Configuration	11%	17%
Reconfiguration Policy (Prototyping)	28%	37%
Reconfiguration Interface	68%	41%
Conditions	83%	87%

Table 1. *Effort compared to high-level language implementation.*

Effort

We focused on the effort required to implement requirements changes. Our experience with the military domain shows that the requirements will change so many times that the effort spent to modify the compensating reconfiguration code will overshadow the initial construction effort. In addition, the ease of implementing changes makes for an effective prototyping tool. The effort metrics we used were source lines of code (SLOC)[1] and time. The initial effort using Bullpen averaged 84 percent of the effort required to implement the same functional system using only high-level code (Table 1).

The effort required to change the functionality of the compensating reconfiguration software was much less with Bullpen than with a high-level source code approach. Bullpen did not perform as well when the changes were to the System Configuration as it did in the other categories, but these were the simplest changes to implement in both systems.

Complexity

We also examined the complexity of the modifications as a result of the requirements change. We reasoned that less complex code is easier to build and less error prone. We used three metrics to determine complexity: number of locations in the code modified, number of defects found during integration test, and repair time for those defects (which includes all types of defects, regardless of their cause). Our reasoning was that more complex code will tend to produce more defects, and those defects will be more difficult to repair. The complexity of the code used to build the initial systems using Bullpen was only 68 percent of the complexity of the high-level code version.

In the most common categories of requirements change, Bullpen showed the best performance (Table 2). As the composition of the simulation evolved, Bullpen was far less complex to deal with than high-level language. The changes to the conditions category were the worst performers again. In the areas of change most commonly encountered in military simulations, Bullpen far out-performed the conventional approach.

Correct Reconfigurations

We also looked at Bullpen's tendency to produce correct reconfigurations. We define a correct reconfiguration as one that results in all objects in the virtual world being controlled by only one executing simulator. We assume that all the simulators in the distributed simulation have been validated and verified. Therefore, objects under the control of a validated and verified simulator will behave realistically. An object not under control of a functioning simulator is bound to eventually behave unrealistically. Should an object be under the simultaneous control of two simulators, it also is not guaranteed to behave in a realistic manner. We assume that a compensating reconfiguration component that makes more incorrect reconfigurations in integration test is more likely to make incorrect reconfigurations in actual use. Bullpen showed that is it equal to or better than high-level language in all categories (Table 3).

Response Time

Finally, we looked at response time. Abstract interfaces generally impose a performance penalty as a cost of an easier-to-understand interface. For Bullpen to be a worthwhile tool, the performance penalty must be within acceptable limits. Since 100 milliseconds is perceived as instantaneous by humans, we were willing to accept a penalty of about 100 milliseconds. Bullpen-generated code took an average of 102 milliseconds longer to determine the correct reconfiguration. In neither case was the decision time significant with respect to the total response time, which demonstrates that the Bullpen approach is fast enough to be practical.

Conclusions

Through our work in the distributed simulation domain, we believe that it is possible to build self-reconfiguring distributed systems using an abstract interface. The advantages of developing distributed systems this way include less effort, less complicated code, and fewer errors. A rule-based abstract interface is powerful enough to handle the reconfiguration requirements found in the military distributed simulation domain. In addition, the response time is fast enough for virtual simulations.

With Bullpen, we can build compensating reconfiguration components with less initial effort, but more important, the code is less complex and easier to modify in response to changing requirements. We found that changes to requirements that involve the virtual world, both the virtual world and system configuration, the reconfiguration policy, and the reconfiguration interface showed the greatest gains

Table 2. *Categories of change compared to high-level language implementation.*

Change Category	Locations	Defects	Repair Time
Virtual Configuration (User Driven)	21%	17%	7%
System Configuration (Prototyping)	63%	100%	100%
Virtual Configuration and System Configuration	25%	100%	100%
Reconfiguration Policy (Prototyping)	19%	14%	8%
Reconfiguration Interface	32%	14%	13%
Conditions	125%	100%	100%

Change Category	Average Incorrect Reconfigurations
Virtual Configuration (User Driven)	25%
System Configuration (Prototyping)	100%
Virtual Configuration and System Configuration	100%
Reconfiguration Policy (Prototyping)	11%
Reconfiguration Interface	27%
Conditions	100%

Table 3. *Categories of change and performance compared to high-level language implementation.*

using Bullpen. These categories also represent the most common types of requirements changes that occur in the military distributed system domain.

The effort and complexity saving shown by this approach supports prototyping. Experimenting with different mixes of spare resources and reconfiguration policy should allow the builders to achieve more effective self-reconfiguring code. Our approach lowers the cost of this experimentation.

We found that our system was more than powerful enough to handle the requirements of military distributed simulations; therefore, we believe that this approach will generalize to other distributed systems that must reconfigure themselves during execution in response to changing conditions. Even though our initial work has been with the compensating reconfiguration function as a component of the distributed program, we believe the proper place for compensating reconfiguration is in the middleware. ◆

About the Authors

Lt. Col. Don Welch is an associate professor of computer science at the U.S. Military Academy (USMA). He teaches software engineering and has experience as an Army software engineer. His military assignments include infantry and special missions units. His current research interests include dynamic reconfiguration, software engineering, managing the risk from year 2000 failures, and distributed simulation. He has a bachelor's degree from USMA, a master's degree in computer science from California Polytechnic State University, and a doctorate from the University of Maryland.

Department of Electrical Engineering and Computer Science
United States Military Academy
West Point, NY 10996
E-mail: Donald-Welch@usma.edu

James Purtilo is an associate professor of computer science at the University of Maryland, where he also holds an appointment in the Institute for Advanced Computer Studies. He is a senior member of the Institute of Electrical and Electronics Engineers, having previously received a doctorate from the University of Illinois at Urbana. His research is in software engineering, with a special focus on software interconnection. Most recently, he served as general chairman for the Fourth International Conference on Configurable Distributed Systems.

Department of Computer Science
University of Maryland
College Park, MD 20741
E-mail: purtilo@cs.umd.edu

References

1. Calvin, J. and D. Van Hook, "Agents: An Architectural Construct to Support Distributed Simulation," *Proceedings of the 11th Distributed Interactive Simulation Standards Workshop*, September 1994.
2. Weatherly, R., A. Wilson, B. Canova, E. Page, A. Zabek, and M. Fischer, "Advanced Distributed Simulation Through the Aggregate-Level Simulation Protocol," *29th International Conference on System Sciences*, Wailea, Hawaii, Jan. 3-6, 1996, pp. 407-415.
3. Defense Modeling and Simulation Office, *High-Level Architecture Rules*, Version 1.2, August 1997.
4. Defense Modeling and Simulation Office, *High-Level Architecture Interface Specification*, Version 1.2, August 1997.
5. Defense Modeling and Simulation Office, *High-Level Architecture Object Model Template*, Version 1.1, February 1997.
6. Siegel, J., *CORBA Fundamentals and Programming*, Wiley Computer Publishing Group, New York, 1996.
7. Purtilo, J., "The Polylith Software Bus," *ACM Transactions on Programming Languages*, Vol. 16, January 1994, pp. 151-174.
8. Kramer, J. and J. Magee, "The Evolving Philosopher's Problem: Dynamic Change Management," *IEEE Transactions on Software Engineering*, Vol. 16, No. 11, November 1990, pp. 1293-1306.
9. Hofmeister, C. and J. Purtilo, "Dynamic Reconfiguration of Distributed Programs," *Proceedings of the 11th International Conference on Distributed Computing Systems*, 1991, pp. 560-571.
10. Minsky, N., "Independent On-Line Monitoring of Evolving Systems," *Proceedings of the 18th International Conference on Software Engineering*," March 1996.

Note

1. Size includes the number of SLOC added, modified, and removed. If code was made "dead" or nonreachable by other modifications, it was removed and not counted.

Real-World Java Development Experiences

A Background Data Collection System

Jerome B. Soller, James Clingenpeel, Patrick W. Hayes Jr.,
Mark Muday, Brian Larsen, and Tamara Jones
CogniTech Corporation

An Internet-based infrastructure can enable easier data entry, validation, and report generation capabilities when users are at different locations or use diverse hardware platforms and operating systems. However, there are many technical obstacles to tackle before such a solution can be successfully implemented, especially when users with diverse requirements need access to extensive capabilities and dynamic information. This article discusses the development and run-time implementation of a background data collection system, based on a three-tier architecture using Java™, the eXtensible Markup Language (XML), and relational database technologies. This solution is representative of a broader class of solutions, applicable to problems requiring the collection, verification, auditing, and reporting of data for users at different locations or who use different hardware and software platforms.

One of CogniTech's customers collects and verifies background data on various applicants to determine their suitability to perform specified tasks as a contractor or an employee for other organizations (clients). The background data include education, professional certification, employment history, criminal allegations, and citizenship. The business processes of collecting and verifying these data are time- and resource-intensive and require the centralized storage and maintenance of the resulting records.

To increase the efficiency, responsiveness, and quality of its verification services, the customer wanted to reengineer its business processes through the use of software packages for each of the three distinct but interdependent services:

- Collection of background data.
- Verification and auditing to authenticate the data.
- Generation of reports and other forms of data access based upon the verified data.

The profile of each applicant determines the data collected, verified, and reported during the process. Therefore, each applicant requires a customized user interface, which is determined dynamically, based on the clients requesting the collection and verification services. CogniTech developed an Internet solution to support the collection of the background data.

The Development of Internet-Based Data Collection System

User Perspective and Involvement

Typical users of the Internet-based data collection are applicants. Under the traditional form system, an applicant might be required to complete as many as 15 forms for different clients, many of which request the same data. From the point that the paper form is sent to the applicant, there is typically a two-week delay until the completed information is returned by mail. This translates into a two-week delay before the customer's verification stage can begin. The transcription of this information from the paper form to a computerized system can lead to incorrect copying of handwriting, a lack of consistent terminology, and inefficiencies within the verification process. To help prepare a precise set of functional requirements, two categories of domain experts—applicants and verification process domain experts—assisted in the development process.

Data Model

A common data model served as the foundation for all of the software solutions. The data model was based on input from domain experts and existing paper data collection forms associated with clients. This model divided the data into two categories: generic to all clients and client-specific. When the valid values for a field could be enumerated, these values were coded with a standard internal representation and vocabulary for presentation to the user.

Technology Selection and Challenges

Decision to Support Internet Data Entry. The first technology decision for the software development was the use of an Internet interface for data entry. This allowed centralized storage and maintenance of the software, databases, and data. It required a minimal information systems investment (a browser and an Internet connection) from its users.

Selection of Programming Language and Target Platforms. Common Gateway Interface (CGI) was the first technology considered for this project. CGI scripts running on a server can dynamically generate HyperText Markup Language (HTML) pages. The advantage of this approach is that it does not require special browser support. Even with the addition of JavaScript (a scripting language for HTML pages unrelated to the Java programming language), static HTML could not capture the dynamic

behavior required by the functional specifications.

In addition to dynamic behavior, the user interface specifications required a minimal security risk, cross-platform support, software reuse, and rapid development. The Java programming language was the programming language that best met these criteria. We first looked at Java Development Kit (JDK) 1.02, an older version of Java supported by most Web browsers in use, but found that it did not support the necessary infrastructure for the rapid development of business solutions. Instead, we chose JDK 1.1 (a later version of the JDK released in spring 1997) and its Java Beans component model [2], which provided the necessary foundation for the solution. The trade-off was that JDK 1.1-compliant software requires that the user have a recent version of Netscape Navigator, Microsoft Internet Explorer, or another Java Virtual Machine (run-time Java environment).

Selection of the Software Development Environments. Microsoft Windows NT was chosen as the software development platform. After developing and testing the JDK 1.1-compliant Java software on NT, it could later be ported to a more stable, scalable, and secure enterprise-wide production environment, such as a UNIX-based server, an AS/400 server, or a mainframe. For a number of reasons, the CogniTech development team selected IBM VisualAge for Java Enterprise Edition as the primary Java development environment for this project. Among its advantages, VisualAge for Java

- Offered strong support for the platforms and technologies within the target enterprise, including JDK 1.1, Windows NT, AS/400 servers, the IBM DB2 Database, and the Lotus Domino E-mail/groupware server.
- Offered strong functionality for team development, including a development repository and strong revision control.
- Supported the model-view-controller paradigm, which encourages developers to fully utilize the object-oriented paradigm by dividing objects by their functionality.

NetObjects Fusion was chosen as the Web site authoring environment for the complex site, which anchored the Java applets. RoboHelp Office Edition supported the creation of the HTML help pages and their associated index, implemented as a Java applet.

Use of XML. We first looked at the Standard Generalized Markup Language (SGML) [1], an international standard to provide a simple, platform-independent and flexible representation. It also is widely used to encode documents. SGML does not define a set of element types (such as tags or attributes) but provides a mechanism to define markup languages to solve a class of problems. HTML is an application of SGML. We chose to implement the eXtensible Markup Language (XML) [9], a subset of SGML that provides many of the benefits of SGML, such as generalized markups to create customized tags, but removes several complex features.

The software uses XML to represent three distinct forms of business domain content:

- The visual layout of components within the user interface.
- The relationships of the screens within the user interface.
- The mapping of the visual components to external data object references, e.g., relational database data.

An intuitive visual XML editor accelerates the development and maintenance processes.

Run-Time Environment

Three-Tier Architecture

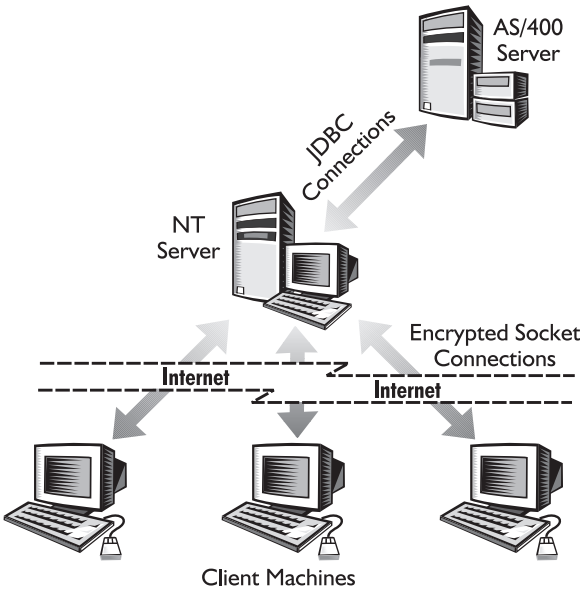
Figure 1 shows the architecture for the solution. The IBM AS/400 server contains the DB2 databases used by the software. A Windows NT server provides a firewall and second tier, which runs the Java implementations of business logic and data access. The XML represents the dynamic mapping to these server-side objects. The Java applets communicate with the server-side Java through sockets, which are low-level communications application programming interfaces (APIs). The client-side Java applets contain the user interface.

Client-Side Java

Since each applicant in the process has a different user profile, the solution generates and dynamically merges content at run-time based on each profile. This results in a user interface that supports a wide variety of navigation paradigms. However, the typical user would only use the system once every two years. Therefore, it is difficult to standardize the choice of client-side Java Virtual Machine for all users.

In theory, one of the advantages of client-side Java is "write once, run anywhere." As the technology is currently supported by many Web browsers and Java Virtual Machines, the reality of cross-platform Java support is "write once, test everywhere." Support for JDK 1.1.2 by Microsoft Internet Explorer and Netscape Navigator has only recently been implemented, and both browsers still contain minor incompatibilities with the standard. Because of the inability to fully control the client platforms used for data entry, the Java applets will run under

Figure 1. *Three-tier architecture.*



Netscape Navigator 4.04 with the Java 1.1 patch, Microsoft Internet Explorer with service pack 1, and fully JDK1.1.2-compliant Java Virtual Machines.

To address the concerns over support for future releases of the JDK, JavaSoft (Sun Microsystem's Java division) recently released a standard plug-in Java Virtual Machine for Web browsers called the Activator. In spite of these "growing pains," Java's cross-platform capabilities have significant advantages over other technologies. In environments where the enterprise controls the client platform, it is significantly easier to upgrade and standardize upon a browser, Lotus Notes client, or Java Virtual Machine than to repeatedly upgrade all computers to new operating systems.

Given customer requirements to support multiple browsers, the solution could not make strong assumptions about locally resident Java class libraries, e.g., Java Foundation Classes. Any classes not resident in the local browser or Java environment would require downloading. To minimize download time, we chose to create a lightweight set of graphical components. Due to incompatibilities in browsers, sockets were used in place of a more robust distributed object framework, such as the Object Management Group's (OMG's) [7] Common Object Request Broker Architecture (CORBA) [8]. This eliminated the need to download the class libraries for the Object Request Broker (ORB) or the Java stubs for the Interface Definition Language (IDL) of a given interface. However, the design supported the possibility of a migration to a CORBA architecture in the future.

Server-Side Java

The server-side Java application on the middle tier of the three-tier architecture handled all database transactions with the third tier AS/400 DB2 database, business logic, and dynamic user profiling. Communication with the database used the Java Database Connectivity (JDBC) [6] standard, which encapsulates Structured Query Language (SQL) statements in Java classes. For Java components only running on the server, run-time cross-platform support is not as important as run-time performance. Java has advantages over C++ because of the ease of developing and porting the software to another server platform.

Lessons Learned and New Technologies

During and after CogniTech's efforts to develop the solution, technologies and developer tools have evolved. Given its experience, the development team formulated a technical road map for future projects.

Most current and future CogniTech projects, including the development of decision support systems and occupational medicine and worker's compensation records, use an object modeling tool, e.g., Rational Rose, and a data modeling tool, e.g., Platinum ERwin, for the design phases. Rose can generate Java class definitions and CORBA IDL specifications, which serve as the starting point for the software implementation. The data modeling tools can generate an SQL Data Definition Language (DDL) specification for the database tables.

In the three-tier architecture previously discussed, the JVM, the Web server, and the database server ran as separate processes. The increasingly popular Java application servers offer an alternative for the run-time implementation of Java middle tiers. Java application servers, such as IBM WebSphere, increase the performance of server-side Java software by supporting a tight coupling of Web server HTTP services, CORBA Internet Inter-ORB Protocol, database connectivity, and JVMs.

The server-side Java issues focus on

- Speed performance.
- The ability to port development code.
- The ability to integrate enterprise data and applications.

Unlike client-side Java, a large percentage of server-side Java software requires high performance and does not require the run-time cross-platform support of Java bytecode (the intermediate Java code sent to the client machine). In the past, components with these requirements could be written in C++ and linked to the Java code through a native interface. With the creation of native code Java compilers (such as IBM's high performance Java compiler), high performance code previously written in C or C++ can now be written in Java. This combines the development advantages of Java, such as its lack of pointers, with the performance advantages of C++.

However, whether the native methods are written in Java or C++, they must interact with cross-platform Java bytecode. Many CogniTech projects will require a high-performance component written in the Java language (such as database queries, user profiling, statistical computations, manipulation of an expert systems knowledge base) to impact the dynamic generation of Java objects, which are then downloaded to many platforms.

A large business issue today is the selection of the appropriate Internet technology to solve a problem. If the user accesses a Web site infrequently, there may be limited incentive to upgrade or standardize their browser or JVM. When the user interface requirements are not complex, CogniTech recommends a dynamically generated HTML and JavaScript user interface.

Servlets [5] and Java Server Pages [4] offer Java programmers a high-performance alternative to CGI for dynamic server-generated HTML. Servlet engines provide a high-performance plug-in that runs a Java Virtual Machine and maintains data values associated with a session. The IBM WebSphere Java Application Server includes a servlet engine and manages database connections. Servlet design and maintenance benefit from the separation of business logic and presentation logic.

The Java Server Pages technology supports HTML produced from a variety of sources, including Web authoring tools. Java Server Pages allow the substitution of HTML within print statements and the combination of Java expressions with HTML tags and content.

If the user interface requires the richness of Java, the organization associated with the Web site must create a strong incentive for users to upgrade their browser or JVM. It is easier to require this standardization if the user

accesses the same Internet or enterprise Intranet site repeatedly. Given this usage pattern, there is a strong business case for supporting the most recent JDK and providing the richest possible Java user interface. In these circumstances, CogniTech recommends that its clients standardize either on the Lotus Notes 5.0 client or a Web browser with the Sun Java Activator plug-in.

For many current projects, CogniTech is developing two user interfaces:

- An HTML and JavaScript user interface generated by Java Server Pages.
- A rich Java user interface, using the Java Foundation Classes [3].

In the solution described above, the developers chose not to use CORBA. However, several technology changes will impact the future use of Java and CORBA. The upcoming JDK 1.2.x will include a lightweight CORBA ORB and components of the CORBA security service. This eliminates the need to download the ORB for browsers using JDK 1.2. For browsers not supporting 1.2, the OMG is releasing a specification for minimum CORBA, which defines a minimum set of CORBA services. This specification could standardize the minimum download of a CORBA ORB.

The CORBA standards offer many technical advantages over Microsoft's competing Component Object Model (COM). However, one significant advantage of COM is that Microsoft bundles it free within the operating system. We believe the greatest challenges of CORBA technology vendors are

- Decreasing the cost of CORBA development tools, ORBs, CORBA services, and application servers.
- Improving the usability of software development environment support of CORBA, e.g., hiding the complexity of CORBA from developers.
- Continuing the development of vertical industry interface standards.
- Developing a business object facility.

Security of confidential information passed across the Internet is another rapidly evolving area. Technology and licensing issues impact the use of security in Java applications and applets. Today, implementations of Secure Sockets Layer (SSL) use RSA encryption. Although public domain implementations of RSA encryption exist, RSA's patent requires a licensing fee in the United States. From an independent developer perspective, licenses for applet downloads of the RSA encryption technology are expensive. This has hampered developer use of these technologies with Java applets. Today's options for Internet developers who do not wish to purchase the RSA licenses include

- The use of other encryption algorithms.
- Creating only HTML user interfaces, using HTTP as the protocol, and leveraging the SSL support of the browsers and Web servers.
- Using CORBA IIOP as the protocol and SSL services provided by several ORB vendors.

Several future occurrences will impact the use of encryption technologies on the Internet. In the year 2000, RSA's patent expires, and RSA encryption becomes effectively free. The Internet Engineering Task Force (IETF) developed a public key infrastructure (PKIX) standard draft, and IBM has released the source code for a PKIX reference implementation to the public domain.

Conclusion

The data collection solution described in this article represents a large class of problems that benefit from a three-tier architecture leveraging Java, XML, and relational database technologies. It is important to understand the current limitations and projected capabilities of these technologies to meet project deadlines and create software components, which can be reused and maintained over time. ◆

About the Authors

Jerome B. Soller is president of CogniTech Corporation in Salt Lake City, Utah, which provides information systems research, design, development, and consulting services. He has a bachelor's degree in electrical engineering from the Johns Hopkins University and holds a doctorate in computer science from the University of Utah. He previously served as a research instructor at the University of Utah and held several research positions for the Department of Veterans Affairs and other organizations.

CogniTech Corporation
1060 East 100 South #202
Salt Lake City, UT 84102
Voice: 801-322-0101
Fax: 801-322-0975
E-mail: Info@Cognitech-UT.com
Internet: http://www.cognitech-ut.com

James Clingenpeel, Patrick W. Hayes Jr., Mark Muday, Brian Larsen, and Tamara Jones are software engineers at CogniTech Corporation.

References

1. International Organization for Standardization (ISO), "The Standard Generalized Markup Language (SGML)," *ISO 8879:1986*, 1986.
2. JavaSoft, "JavaBeans FAQ: General Questions," http://www.javasoft.com/beans/faq/faq.general.html, 1998.
3. JavaSoft, "Java Foundation Classes: Now and the Future," http://www.javasoft.com/marketing/collateral/foundation_classes.html, 1998.
4. JavaSoft, "Java Server Pages: The Front Door to Enterprise Applications," http://www.javasoft.com/products/jsp/index.html, 1998.
5. JavaSoft, "Servlet," http://www.javasoft.com/products/jdk/1.2/cast-out/servlet/index.html,1998.
6. JavaSoft, "The JDBC Database Access API," http://www.javasoft.com/products/jdbc/index.html, 1998.
7. Object Management Group, "OMG Home Page," http://www.omg.org, 1998.
8. Object Management Group, "OMG Technical Library," http://www.omg.org/library/downinst.html, 1998.
9. World Wide Web Consortium, "Extensible Markup Language (XML)," http://www.w3.org/XML/,1998.

Outsourcing and Privatizing Information Technology

Re-examining the "Savings"

J. Michael Brower
Department of Justice, Immigration, and Naturalization Service

The Department of Defense has taken about 80 percent of the government cutbacks since the end of the Cold War. As a result, information technology (IT) outsourcing and privatization has become a popular means to lower the cost of labor devoted to perform computer-related functions. This article advances a labor theory of value to explain the source of the profit and cost savings that underwrite outsourcing and privatization as popular financial tactics. This labor theory uses several private and public industry examples to promote its thesis and also explains the impact of U.S. industry's use of foreign programmers to reduce or cap wages. A case is made that outsourcing and privatization undermine long-term economic stability, ultimately weakening national security institutions dependant on IT.

Cutback-stricken employees in both the public and the private sector are worried, and rightly so, about information technology outsourcing and privatization (ITO&P). The problem has reached its apex at the Department of Defense (DoD) where many government technologists face the prospect of contractorization as military budgets become more austere and personnel cuts increasingly severe.

Since the Cold War's end, DoD has borne about 80 percent of all government cutbacks, which, after four rounds of base closures, have resulted in the loss of approximately 355,000 civilian and 743,000 military jobs since the early 1990s. This means more competition for private technology workers and less security for mid- and lower-level defense employees who still perform DoD information technology (IT) tasks.

All defense IT workers know jobs are made and lost after leaders embrace or rebuke ITO&P. But they wonder about the financial mechanics and the art of and their part in this big financial deal. Nowhere are the battle lines more apparent and consequential to DoD's future than in the struggle between public and private employees for government IT work.

The heavy cuts in DoD's permanent work force have still failed to generate savings enough to offset planned procurement expenditures called for under the May 1997 Quadrennial Defense Review (QDR). In June 1998, 15 business leaders from the Business Executives for National Security (BENS), a group of U.S. defense contractor executives, declared that DoD could make up the procurement shortfall of around $15 billion through more aggressive outsourcing. John Lis, a BENS policy associate, in a recent issue of *Defense News* suggests that more contracting-out in payroll, utilities, information systems, housing, and other base support functions would save billions of dollars. DoD's 1998 procurement budget is $45 billion, while it requires $60 billion to fund "all of its authorized procurement programs," according to the article [1].

We can conclude with confidence from this that programming, networking, webmaster, and other IT skills will continue to migrate from DoD to the public sector, at least in the short term. And this will occur despite the actions of some agencies like NASA to federalize private contractors at high grades to keep them focused on intra-agency IT projects.

Outsourcing and Privatization — Differences and Commonalities

Outsourcing

This brand of economic determinism is a form of contracting-out that promises a satisfactory level of accuracy, quality, timeliness, etc., while shunting native talent to core tasks. Outsourcing-type contracts can be government-to-government, government-to-private, or private-to-private arrangements.[1] It is mainly the potential of reducing labor costs that compels many a chief information officer, entrepreneur, or government executive to tinker with outsourcing.

Privatization

Economist Calvin A. Kent's still-timely definition in *Entrepreneurship and the Privatizing of Government* tells us that privatization "refers to the transfer of functions previously performed exclusively by government, usually at zero or below full-cost prices, to the private sector at prices that clear the market and reflect the full costs of production." [2]

In a shortsighted pursuit of profits, government employees are often *right-out-the-door-sized* from secure positions and flung into the private sector, often flooding a given labor market, flattening wages, and forcing public and private sector employees into intense competition for limited positions. In the last analysis, it is precisely this competition that lowers the cost of available labor and thus provides the lion's share of company outsourcing profits.

Reducing the costs of labor for competent code writers, systems integrators, and information specialists—in a word, every genre of technologist—is the

heart and soul of any success that ITO&P can claim for stakeholders and shareholders.

Outsourcing Information Technology – IT Is a Small World

One of the bellwether battles over public vs. private work is being fought in the technology arena. *Federal Computer Week* reported on June 8, 1998 that the Office of Management and Budget is pushing for a new list of government activities that might be outsourced. Similarly, industry has been lobbying hard to revamp OMB Circular A-76, the federal guidebook covering public vs. private competitions for work, to expand and strengthen enforcement options. Edward DeSeve, acting deputy director for management at OMB, speaking at a recent conference sponsored by the Professional Services Council, explained that DoD has outsourced 150,000 full-time positions and saved $6.4 billion in the process [3]. Obviously, the trend toward outsourcing is continuing, mission-related IT requirements are increasing, and government staff continues to decline. In the last analysis, capitalizing on less expensive labor makes ITO&P pay.

The hocus-pocus at work behind the recent "job creating" move in Pennsylvania to outsource mainframe work will not hold up under close scrutiny by those technologists who understand outsourcing's true source of value—reducing the labor expense of technologists! Pennsylvania's state government officials hail the move to consolidate and outsource work performed by 23 state data centers. According to *Government Computer News/State & Local*, Pennsylvania's well-meaning but uninformed officials claim savings of $127 million over the next five years and $25 million annually for the other two years of the contract [4]. The contract is worth about a half billion dollars over its life. Only at the end of the article does one find the human cost: "The commonwealth will encourage the contractor to hire the 370 data center employees who will be displaced by the new setup. …"—this is what it is all about. Butchering the decent salaries of the current and necessarily voiceless IT workers will offset any losses that the outsourcing strategy might otherwise pass on to the state—yes, they will show a profit. The displaced will seek—quietly—acceptance in the new company that gets the outsourced work. Even those who receive higher wages in the short term will sacrifice their health benefits, their state retirement options, their decent working hours, and job security. The economic analysis behind the venture—in this case conducted by well-paid and well-known industry supporter KPMG Peat Marwick of New York, cannot be expected to concede anything to government IT employees—quite the reverse.

State governments all over America, like the federal government, look to capitalize on perceived savings through privatizing and outsourcing IT. Orchestrated by Chief Information Officer Elizabeth Boatman, a significant part of the privatization effort currently under way in Chicago's city government involves contracting-out departmental IT functions. Targets include arrest process computerization, legal case management, and financial systems. The work is frequently repetitive and labor-intensive, choice breeding grounds for privatization and outsourcing projects. The primary motivation to go private in this case is the lure of short-term cost reductions (the largest expenses are usually personnel-related) through a strategic decision to purchase rather than grow expertise. In an interview with *Government Computer News/State & Local*, Boatman stated, "Privatization has hinged on issues of cost. ...We've also had a difficult time competing with private industry for good IT employees. Retraining the ones we do have is costly, and we simply don't have time to wait for the payoff." In general, specialized service capabilities, economies of scale, niche advantages, and the flexibility of private employees over public employees factor heavily into the privatization decision. But the greatest attraction remains the potential to reduce labor expenses to achieve management goals. The largest part of the city's IT spending (30 percent of the $64 million calendar year 1998 budget) is devoted to personnel [5]. Boatman has discovered that privatization, like outsourcing, delivers a blacker bottom line via the "pink-slip approach." Interestingly, professional service companies are beginning to admit that they do not know if customers are saving money, breaking even, or expending more to transfer functions out-of-house for the cause of ITO&P. In reality, most companies are "exploring outsourcing services for cost predictability" rather than for assurances that outsourcing costs less [6].

Consider, too, the recent decision by pharmaceutical maker Eli Lilly & Co. to outsource its on-line health-care network to IT giant EDS. Eli Lilly decided to shed undesirably expensive payroll by effectively putting IT workers into the company that receives the outsourced project. Eli Lilly staff who attempt to follow their jobs will look for employment with EDS. Those newly created job seekers will be "offered jobs at EDS based on skills" according to an EDS spokeswoman [7]. It is a short-term, stock-enhancing, win-win for both companies—EDS gets a labor pool hungry for work; Eli Lilly reduces its costly rolls. The unionless (over 85 percent of the U.S. work force) watch their wages stay flat or decrease as they unwittingly saturate the available labor pool.

Another labor-cheapening tactic particularly effective against permanent DoD IT staff is importing foreign IT employees. Roy Beck, editor of *The Social Contract* magazine, has written that instead of training their own cadre of technologists, Microsoft Corp. prefers to "import tens of thousands of foreign programmers" or ship work overseas because wages are lower [8]. In his book, *The Case Against Immigration*, Beck cites the 1990 census, which found foreign-born IT workers in Silicon Valley will work for nearly "$7,000 less than did natives of the same age and level of education." [9] He also reveals that computer and software makers have a willing cooperative accomplice—universities attempt to artificially keep wages low. Beck writes that many institutions of higher learning "have kept their Ph.D. numbers up by

increasingly turning to foreign students. So the universities crank out far more scientists than are needed for industry, the U.S. government, and for university professorships. The glut works further to the universities' advantages because there is a large pool of scientists willing to continue to work for low wages in postdoctoral research positions for another three to six years. The universities, therefore, gain an even larger low-paid work force." [10] This too is the real value of ITO&P outsourcing: a slashed labor cost.

Technically minded immigrant labor, particularly from China, Pakistan, and India, will work for far less than U.S. citizens when a potential green card is part of their employment package. Now that Congress is being told by information systems giants like Intel and Microsoft that foreign IT worker quotas must be increased to assure a flow of new program and network administrators, unorganized American computer specialists will be more amenable to bargaining. The *Wall Street Journal* recently discussed how high-technology companies have asked for an exception to immigrant quota levels to permit more foreign IT workers into the country. Intel's president, Craig Barrett, contends that if federal limits on technical immigrant personnel remain at current levels, "the talent will go where the opportunities are, even if that is offshore." [11] Indeed, *Business Week's* chief economist, William Wolman, similarly discovered with co-author Anne Colamosca in their new book *The Judas Economy* that when capital learns that it can outsource for computer programmers and code writers in Beijing and New Delhi at one-third the wage of similarly skilled U.S. IT workers, stockholders will demand that capital fly to China and India [12]. Not long will the remaining community of civilian and military technologists at DoD have to wait until this strain of outsourcing visits them.

Conclusion

In no department in the federal sector are the ravages of ITO&P more apparent than in DoD. Many thousands of defense industry and government IT workers worldwide have suffered the effects of blind contracting-out, outsourcing, and privatization. When the vogue of outsourcing and privatization fades away, its legacy will be one of short-term fiscal advantage, long-term economic instability, and ultimately a weakening of national security institutions—all at the expense of the average technology workers from whom ITO&P draws its chief source of value and profit. ◆

About the Author

J. Michael Brower is a program specialist with the Department of Justice, Immigration and Naturalization Service in South Burlington, Vt. He was assigned to the Army secretariat from 1991 to 1997 with the Information Management Support Center. His previous assignment was with the Office of the Assistant Secretary of the Army (Financial Management and Comptroller), Business Practices Directorate at the Pentagon. He has a bachelor's degree in business management and has published many articles on military, information management, privatization, and outsourcing issues in numerous journals such as Armed Forces Journal International, Program Manager, Minerva, Stars and Stripes (domestic edition), and the Army News Service.

E-mail: jmichael@together.net or john.m.brower@usdoj.gov.
Internet: http://www.geocities.com/capitolhill/lobby/2985/

References

1. Burgess, Lisa, "Business Leaders Promote Outsourcing to Pentagon," *Defense News*, June 8-14, 1998, p. 44.
2. Kent, Calvin A., ed., *Entrepreneurship and the Privatizing of Government*, Quorum Books, New York, 1987, p. 4.
3. Tillett, Scott, "OMB: List Outsource Possibilities," *Federal Computer Week*, June 1998, p. 10.
4. House, Claire E., "Pennsylvania Takes Bids to Outsource Mainframe Work," *Government Computer News/State & Local*, September 1998, p. 1.
5. McCann, Michelle, "Chicago Privatizes Some Systems Work; Statistic from Chicago's Business Information Systems Department," *Government Computer News/State & Local*, February 1998, p. 8.
6. Bowen, Ted S. and Martin LaMonica, "IT Gets Picky with Outsourcing," *Infoworld*, Aug. 17, 1998, p. 43.
7. McGee, Marianne K. and Gregory Dalton, "Lilly Outsources to EDS," *Information Week*, Feb. 23, 1998, p. 34.
8. Beck, Roy, *The Case Against Immigration*, W. W. Norton & Company, 1996, p. 142.
9. ibid., p. 143.
10. ibid., p. 150.
11. Takahashi, Dean, "Ethnic Networks Help Immigrants Rise in Silicon Valley," *Wall Street Journal*, March 18, 1998, p. B1.
12. Wolman, W. and Anne Colamosca, *The Judas Economy: The Triumph of Capital and the Betrayal of Work*, Addison-Wesley, 1997, pp. 120-154.

Note

1. See Daniel Minoli's *Analyzing Outsourcing*, McGraw-Hill, 1995. This is one of the few texts available the rely heavily on mathematical models. It is an indispensable text for those interested in further study of outsourcing particularly as it impacts IT. See review of this work in the *Army RD&A* magazine, May-June 1998, p. 45.

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 publisher@stsc1.hill.af.mil
Managing Editor	***Forrest Brown*** 801-777-9239 DSN 777-9239 managing_editor@stsc1.hill.af.mil
Senior Editor	***Sandi Gaskin*** 801-777-9722 DSN 777-9722 senior_editor@stsc1.hill.af.mil
Graphics and Design	***Kent Hepworth*** 801-775-5798 graphics@stsc1.hill.af.mil
Associate Editor	***Lorin J. May*** 801-775-5799 backtalk@stsc1.hill.af.mil
Editorial Assistant	***Bonnie May*** 801-777-8045 editorial_assistant@stsc1.hill.af.mil
Features Coordinator	***Denise Sagel*** 801-775-5555 features@stsc1.hill.af.mil
Customer Service	801-775-5555 custserv@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
ESIP On-Line	http://www.esip.hill.af.mil

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: 801-775-5555
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: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request 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. Call 801-777-7026 or DSN 777-7026 for assistance, or E-mail to schreifr@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 Automated Printing Service, Hill AFB, UT 84056. *CROSSTALK* is distributed without charge to individuals actively involved in the defense software development process.

Need Proof? Call 1-900-Y2K-SHAM

A year ago I shook up the technology world with a crushing rebuttal to the paranoia of year 2000 (Y2K) alarmists, assuring our readers that "MacGyver alone could fix the Y2K problem using nothing but PVC pipe, a transistor radio, and Gouda cheese" (http://www.stsc.hill.af.mil/CrossTalk/1998/jan/backtalk.html). Unfortunately, a startling percentage of readers thought I was joking, or at least tried to dismiss my forecast by

- claiming to have proof that MacGyver is a fictional character. (So what is he—a trained otter in a guy suit?)
- claiming that the problem isn't a lack of solutions but a lack of time. (I'll partially concede this point. Cheese factories can only work so fast.)

It's getting harder to be a voice of sanity while virtually all of government and industry—blinded by facts and data—jumps on the "this-could-get-ugly-if-we-don't-act-fast" bandwagon. They're forecasting irritating to catastrophic effects depending on what gets done this year, and the sad thing is they're basing all this alarm on a laughably faulty assumption. All along, they've presumed that just because their critical systems go kablooey when they *test* for Y2K problems, it means these systems *will* go kablooey in January 2000.

Ha! These people are obviously putting more trust in their experience and in the "experts" than in the opinion polls, which clearly show that most people think the Y2K thing is overblown. And the Y2K alarmists aren't the only ones with respectability and expertise on their side, either. After some searching, I have finally found someone with the respectability, experience, and credentials to reliably verify that we'll be buying fresh bananas by credit card come next January:

Consultant: Thank you for calling 1-900-ASTROTEK, the psychic technology hot line. We employ only MCSE-certified astrologers, just $3.95 a minute. Birth date?

Me: May eighth. Tell me: what will January 2000 bring?

C: Let's see *(sound of shuffling)* ... the cards say ... beware of mysterious floppies, for they may foul the purity of your system with macro viruses. Backup often and–

M: –No, I mean, will the Y2K bug bring chaos and upheaval or just overcooked microwave popcorn?

C: For that I must gaze at the crystal ball ... I type January 15, 2000 into the mysterious oracle's keypad and I see ... I see ... *(sound of whacking)* ... blackness ... nothingness. My system just froze up.

M: I have no time for that. Just tell me whether I'll be cooking toothpaste over a can of Sterno next January or whether I'll be dining on imported Argentine beef while making satellite phone calls from a 747.

C: *(shuffling)* The cards say that the answer ... lies within each person's heart.

M: Give me an answer I can work with. Ed Yardeni's heart says there will be a severe worldwide recession and a huge dip in the stock market. Ed Yourdon's heart told him to move into a solar-powered house in New Mexico–

C: –Who are these people?

M: A world-renowned economist and a world-renowned systems consultant.

C: And I couldn't help but notice the similarities in their names. Are you certain they're not the same person?

M: Nobody knows for sure. But they're huge in their fields. Why should I believe my heart ahead of theirs?

C: Does your heart say there will be a Y2K disaster?

M: Well, having seen "Armageddon" and "Deep Impact," there's a good chance Manhattan and Paris will be destroyed. But I know that somewhere out there a group of scrappy misfits will save the planet with seconds to spare, followed by celebratory vignettes of cheering people who represent the Earth's major cultures and ethnic groups.

C: Then, there you have it. That is your truth, and you may live accordingly.

M: So I don't personally need to do anything, just in case someone else's truth happens? What do the stars say?

C: Let's see ... you're a Taurus ... January 2000 ... emphasis on romancxe and hardware upgrades ... 10X DVD drives will be big ... beware of Capricorn network administrators ... nope, nothing about distribution problems, power outages, or hardships. Yes, you'll be ordering Alaskan salmon over the Internet next January.

M: Great! Well, I'm off to sell my food storage to some emergency preparedness nuts and buy some snowmobiles!

What a relief that interview was! But if you're still not sure if you should personally do anything, here's an analogy to put things in perspective: Imagine some weather experts say a hurricane may or may not hit your city on a certain day, and it may or may not have a devastating impact. What's the intelligent thing to do? That's right: you sit on your hiney until general panic sets in, and then you sell bottled water and toilet paper to desperate people at obscene prices. At least that's what I'm going to do. Snowmobiles aren't cheap, you know. – Lorin May

Got an idea for BACKTALK? Send an E-mail to backtalk@stsc1.hill.af.mil