Measurement Is Nothing New

Elizabeth C. L. Starrett (Guest Writer)
Software Technology Support Center

Most people use measurements and metrics every day. They are so ingrained in our normal way of doing things that we often do not realize we are using them. When we drive our car, we watch the gauges, especially the gas, to make sure everything is all right. When we prepare for a vacation and a long drive, we may check the oil and tire pressure to make sure we are not going to have any problems.

While following a recipe, we measure the ingredients and make sure the temperature of the oven is within the correct thresholds.

Women expecting babies go to their doctor on a regular basis to monitor how things are progressing. My doctor performs an ultrasound on me every month. During the ultrasound, he measures the length of the femur, the diameter of the head, and the diameter of the belly. He uses all of these measurements to ensure the baby is progressing as expected. The medical staff also checks my weight each visit to determine if I am gaining the right amount of weight. There are also blood tests taken at periodic intervals to check for other problems (Downs syndrome, diabetes, etc.) and prepare for them.

We even measure how the day is progressing. How many of you are wearing a watch right now? We measure how the day is progressing and plan our actions accordingly. Have you called your spouse to say that you will be later than usual?

Do you make a list of necessities before going to the store so you don't spend more money than you wanted? Have you set a budget?

In the long run, software measurements will make our lives easier. Monitoring our progress on a project lets us know when we need to begin corrective action. It also lets us know if things are progressing better than expected, so there is room for experimenting with potential improvements. We set thresholds in advance to aid with determining when those actions should be taken. Measurements are not a cure-all. We may still find problems that cannot be fixed with advance warning, but knowing about these problems before delivery will help the developer and the customer decide together how to best deal with them.

I was wondering why so many software measurement efforts fail when measurement is so much a part of our everyday life. As I was trying to tie the two concepts together and determine why there would be a difference, it occurred to me that there is no difference. How many times do people get stranded along the road because they did not check their gas gauge? Maybe they were checking it but failed to heed its warning. Have you ever seen signs that your project was in trouble but just held onto the belief that you would remedy things down the road? How many people do not follow a recipe, and it turns out awful? How many people are afraid to go to the doctor because they think there might be something wrong and do not want to know? How many people set budgets and overspend anyway? Are you consistently $50 over budget every month but continue to believe you will spend less next month?

Perhaps the most frustrating situations are those in which people believe they will get different results without changing what they do. Deciding to implement and then institutionalize a measurement system seems essential to improvement. All of the old clichés apply. That is how they got to be old clichés. "If you do not know where you are, a map will not help" or "That which you cannot measure you cannot manage." Every organization, and perhaps every person, needs to begin to measure and then pay attention to what the data is telling them. I, for one, have never enjoyed the highway in the desert on foot. ◆

Free Pizza for Ada Bugs

In the May 1998 issue of *CROSSTALK*, there is a reprint of an Executive Order titled, "Year 2000 Conversion." I am writing to ask *CROSSTALK* readers if any of you have found an Ada Year 2000 bug, and if so, how much effort did it take to correct? I offer free pizza to the first three bug finders responding to this request who show up for a San Diego SigAda meeting prior to the year 2000.

Robert C. Leif, Chairman
ACM San Diego SigAda
Voice and Fax: 619-582-0437
E-mail: rleif@rleif.com

CROSSTALK Useful Classroom Resource

We enjoy your magazine. It is handed out as a resource to every Systems Automation Course student in the software engineering block of instruction, and many of them research further topics they find in your articles.

Avery McConnell
U.S. Army Computer Science School
Fort Gordon, Ga.

The Operational Effects of the Year 2000 Problem

Lt. Col. Scott B. Dufaud
U.S. Air Force Year 2000 Program Management Office

Widespread year 2000 problems in military and civilian computer systems have the potential to open the door to grave consequences for the military and the nation's security in general. To prevent giving our enemies advantage over us in the face of potential chaos, people working in all levels in the nation's defense infrastructure must understand the nature of the risks and to fully participate in plans that ensure no weak links are left exposed.

According to retired Col. John A. Warden III (U.S. Air Force), leadership is the key to success or failure in war, and as such, every action taken in a war should be geared to directly or indirectly affect the enemy's leadership [1]. Given the phenomenal success of Warden's "Instant Thunder" plan employed in the Gulf War, this theory has been proven in battle.

Warden's basic premise is that modern nation-states exist as a "system" that consists of five concentric rings or "centers of gravity." The innermost ring is "leadership" and (moving outward) is followed by "key production," "infrastructure," "population," and finally the outermost ring is "fielded military forces." Because the nation-state operates as a system, each of the centers of gravity is dependent on the others for the survival of the system. Each center of gravity is also directly or indirectly affected by the health and status of the others.

Warden asserts that leadership is the key to success or failure in war because when a nation has endured enough pain inflicted through conflict, the leader will sue for peace or lose power. In other words, the cost for continued resistance and conflict is greater than the consequences faced by laying down arms. He also states that prior to the ascendancy of air power, the only way to directly influence a nation-state's leadership was to first engage and destroy the enemy's fielded military forces. Only then were the other centers of gravity exposed and vulnerable. With advanced technology and superior air power, Warden argues that all aspects of a nation-state are equally vulnerable to attack and destruction from the onset of hostilities [1].

Influence Through Strategic Paralysis

Warden and his team of planners devised the Instant Thunder air campaign to achieve a specific effect called "strategic paralysis" to strike at the heart of the enemy's leadership and bring about a quick end to the conflict. Strategic paralysis explains the effects of disconnecting a nation's leadership from its people and the fielded military forces. This would cause a systemic breakdown of strategically critical functions such as communications, electricity, distribution, and other aspects of the national infrastructure. Instant Thunder successfully attacked Iraq's national leadership by cutting off Saddam Hussein's ability to communicate with his subordinates and by halting the availability of key production and infrastructure facilities.

Because the United States is the most technically dominant nation in the world, we have the distinct advantage and capability to deliver strategic paralysis on our enemies while remaining relatively invulnerable to a reciprocal attack on our nation. However, by using advanced technologies, we have become extremely dependent on those technologies to perform our missions. Every aspect of our society and our military is computerized or automated and therefore relies on immediate access to accurate information. Our reliance on technologies has created built-in vulnerabilities and threats to our ability to carry out the mission. We do not have to worry about strategic paralysis being inflicted upon us by our enemies, but our dependence on technology has exposed us to the year 2000 (Y2K) "time bomb," which has the potential to inflict strategic paralysis from within.

Are We Susceptible to Parallel War?

If the eyes and ears of our nation's leaders are blinded because of the Y2K problem, our enemies will have the opportunity to exert influence on U.S. leadership. Without exposing a single asset or suffering a single casualty of their own, our enemies will enjoy the benefits of a direct attack on the U.S. national leadership in the first ring of Warden's model.

Simple analysis shows that the insidious nature of the Y2K problem on an information-based society will attack all five rings of Warden's model equally and simultaneously. Warden points out that because the concept of "parallel war" brings many parts of a nation's system under simultaneous attack, the system cannot react to defend or to repair itself [2]. Not even the greatest military planners with unlimited military resources could ever accomplish what could happen to our nation-state as a result of the Y2K bomb. Our enemies are watching our progress on fixing what is potentially one of our nation's most significant threats in history.

History is full of examples in which the difference between success and failure in conflict has been determined by the possession or absence of key information at key points in time. Richard Gabriel declares that the one constant in the Mayaguez rescue, the Iran raid, the Lebanon incursion, and the invasion of Grenada was intelligence failure [3]. The intelligence mission and every other military mission we perform is based on getting the right information to the right place at the right time. If we cannot move accurate data in a timely manager because of the Y2K problem, we put our mission success at high risk. Likewise,

Notice of Cancellation

Military Standard Software Development and Documentation

MIL-STD-498 Notice 1 May 27, 1998

MIL-STD-498, dated 5 December 1994, is hereby canceled. Information regarding software development and documentation is now contained in the Institute of Electrical and Electronics Engineers (IEEE)/Electronics Industries Association (EIA) standard, IEEE/EIA 12207, "Information Technology-Software Life Cycle Processes." IEEE/EIA 12207 is packaged in three parts. The three parts are IEEE/EIA 12207.0, "Standard for Information Technology – Software Life Cycle Processes"; "IEEE/EIA 12207.1, "Guide for ISO/IEC 12207, Standard for Information Technology – Software Life Cycle Process – Life Cycle Data"; and IEEE/EIA 12207.2, "Guide for ISO/IEC 12207, Standard for Information Technology – Software Life Cycle Processes – Implementation Considerations."

[DoD activities may obtain copies of the IEEE/EIA standards from the Standardization Order Desk, 700 Robbins Avenue, Building 4/D, Philadelphia, PA 19111-5094. The private sector and other government agencies may purchase copies from the Institute of Electrical and Electronics Engineers, Inc., IEEE Service Center, 445 Hoes Lane, P.O. Box 1331 Piscataway, NJ 08855-1331.]

history also shows us that the availability of accurate and timely intelligence is key to success. This is what enabled Gen. George S. Patton to drive his forces with focused speed: He knew what he could expect from the enemy, where to send fuel and ammunition, and when to shift land and air forces.

Rapid offenses and troop movements are complex and require massive amounts of accurate and timely information [2]. Our entire command and control system is based on our ability to gather, analyze, and disseminate information, all through an "infosphere" that is dependent on technology-based equipment and systems that are vulnerable to the Y2K problem. Our ability to fly hundreds of sorties in a limited airspace is dependent on real-time communication with friendly forces over the Have Quick radio system while denying our enemies the ability to jam or overhear those transmissions. Our ability to detect and assess enemy missile launches depends on satellite hardware and software, communication links, threat analysis software systems, and then communication links to end users. Our ability to launch and complete sorties relies on a multitude of different software and hardware systems: air traffic control, radars, avionics, secure communications, Global Positioning System, mission planning systems and equipment, ordinance avionics, automated test equipment, and simulators, to name a few.

All these systems have two things in common: They process and convey information to the operator, and they are controlled to some degree by *automated* information technology. Not all are "date-aware," but our task is to find out which ones are and to fix them.

What We Have Done to Date

The U.S. Air Force Y2K effort is being carried out by two program management teams, one at the Air Force Communication and Information Center and one at the Air Force Communications Agency. These two teams are supplemented by program offices that reside in each major command (MAJCOM), Field Operating Agency, and Direct Reporting Unit that function as an extension of the headquarters staffs. In addition, the Air Force has fully engaged the functional staffs, assigning responsibility for comprehensive inventories Air Force-wide, researching the compliance information for each item, and sharing this data within their domains to the commanders they support.

There are over 200 primary Y2K points of contact Air Force-wide working full time on this issue. This information and more is found on the Air Force Y2K Web page (http://year2000.af.mil), which is one of the best and most comprehensive resources in the world for information, guidance, and current status of our effort. We have an Internet-hosted on-line, real-time database that provides instant status and access to all the over 3,400 systems we are tracking in the Y2K program. We have created three different guidance packages to direct efforts in the field and have trained over 900 people worldwide in an Air Force-developed and standardized certification process. To sum it up, we have energized the Air Force Y2K effort by mobilizing the support communities, thus ensuring their own domains are squared away for Y2K.

What We Need to Do Now

To date, the communication and information and other support communities have been the "pointy head of the Y2K spear." That is, we are solving the Y2K problem through a process of elimination—systems we are aware of are identified and then systematically renovated through the standard Y2K lifecycle documented in the Air Force Guidance Package.

How can we know we have identified the entire universe of systems—hardware, software, technology-controlled equipment—that the Air Force depends on to complete all our missions? We need to engage the operational communities at every level to leverage their

knowledge of mission processes. This is the only way to guarantee that all our critical missions are free from negative Y2K impacts.

By engaging the operational communities and the systems they employ to carry out wartime operations, we can identify critical mission processes and components previously missed. We need to be working off the commander in chief's designated mission-critical systems listing to ensure that all electronic pathways to and from these systems are Y2K compliant. Because so much of our operational capability is maintained and executed at contingency sites and deployed locations, Y2K vulnerability analysis needs to be performed on the mission processes employed there. MAJCOMs and main operating bases need to ensure that operational planning processes and systems that direct and employ forces at these locations are Y2K ready.

Only through this analysis can we identify the most critical wartime processes and ensure that adequate contingencies have been properly identified and documented. It is time to make the operational mission the pointy head of the spear—we cannot afford to continue looking at the problem from a purely functional perspective. We must widen our scope to look at the entire Air Force as a whole system to find out where we are most vulnerable. The bottom line is that on Jan. 1, 2000, Y2K mission impacts will hinder the commander in chiefs' abilities to perform their missions—it will be too late to do these things that we should be doing now.

The Y2K problem is not just a communication problem—its an Air Force mission problem. The program management office here at Scott Air Force Base encourages everybody to look at their jobs and their units' missions from a Y2K perspective. How will it affect your duties and ability to support the mission? How will if affect your unit's ability to perform its mission? Find out what is being done at your unit and then take appropriate actions to raise issues and contribute to the solution. The Air Force relies on every person so that it can be the greatest air and space force in history; the way we must handle the Y2K problem is no different. Our success depends on having every individual take personal responsibility for Y2K.

Summary

History has proven Warden's theories to be correct. The new paradigm for war in this technology and information-based age is to directly influence the enemy's leadership by affecting his capability to function as a cohesive system. Blind the enemy's leadership by cutting off communications, taint their information or prevent them from receiving it, disrupt key production facilities and other national infrastructure to deflate national morale, and inflict choke points. By denying an enemy any one of these capabilities, an aggressor gains significant advantage. If the Y2K issue is not adequately addressed, we will allow all these things to happen to our National Command Authorities. Our enemies, *all of them*, will achieve these advantages, simultaneously, without any effort on their part. Y2K is the Pearl Harbor of the 21st century just waiting to happen, but only if we let it. ◆

About the Author

Lt. Col. Scott B. Dufaud is deputy program manager for the U.S. Air Force Year 2000 Program Management Office at the Air Force Communications Agency (AFCA), Scott AFB, Ill. Prior to assuming these duties in November 1996, he was the chief of the Software Management Division at AFCA. Dufaud specializes in software management issues, software engineering process groups, software process improvement via the Capability Maturity Model, technology insertion, and issues of accelerating organizational change. He previously served at Headquarters Strategic Air Command and U.S. Strategic Command, the Air Force Manpower and Personnel Center, and Headquarters Air Force Space Command. He has a bachelor's degree in computer science from Southwest Texas State University and a master's degree in systems management from the University of Southern California.

Voice: 618-256-5697 DSN 576-5697
Fax: 618-256-2874 DSN 576-2874
E-mail: scott.dufaud@scott.af.mil
Internet: http://year2000.af.mil

References

1. Reynolds, Col. Richard T., *Heart of the Storm*, Vol. 1, Air University Press, Maxwell Air Force Base, Ala., January 1995, p. 17.
2. Warden III, John A. and Karl P. Magyar, "Air Theory for the Twenty-First Century," *Challenge and Response*, Air University Press, Maxwell Air Force Base, Ala., August 1994, pp. 322-324.
3. Gabriel, Richard A., "Grenada," Air Command and Staff College Seminar and Correspondence Lesson Book 8, Ver. 10, Maxwell Air Force Base, Ala., pp. 32-44.

Year 2000 Certification
Air Force Tenets to Success

Thomas V. Ashton
U.S. Air Force Year 2000 Program Management Office

Certification of your systems is key to surviving year 2000 problems but cannot guarantee success. Certification by definition can add additional liability problems that can plague an organization after the year 2000. The Air Force has created a certification process with seven tenets that if practiced will help guarantee success in the year 2000.

Success in terms of your year 2000 (Y2K) project means not failing in your Y2K fixes. This means they must be done on time—this deadline will not slip. Timing *is* everything, and the time for certification is now.

Why certify? Do you have the time to work for the percent assurance that Jeffrey Voas calls a "utopian pipe dream"? [1] Or would you prefer a confidence based on sound process? Some will do neither and fall prey to the "I hope" syndrome, where no certification is performed and managers rely on normal testing to avoid failing. But the Y2K issue is an addition of a magnitude to the normal software challenge. Peter de Jager points out that we as software managers are late 50 percent of the time [2]. If your system has interfaces, you not only have your concerns but also your suppliers' and users' concerns as well. With all the renovation being done, each change creates another magnitude of problems that require regression testing. These factors, along with the additional Y2K burden, make certification a must.

The Air Force Certification process is built on two Air Force-recognized standards: The Air Force five-phased approach to the Y2K and the Year 2000 Compliance Checklist. These two form a solid foundation for certification. The five-phased approach is further described in the Air Force Year 2000 Implementation Plan (formerly known as the Guidance Package) and was adopted not only by the Air Force but also by the Department of Defense (DoD) and other federal agencies in early 1996. The Compliance Checklist is well known throughout the DoD and is also being accepted by other agencies.

On top of this foundation, forming the structure, is a set of seven basic tenets that bring a higher level of confidence to the certification process from the bottom, where the technical development is performed to the top levels of management:

- Consistency Within the Process.
- Working as a Team.
- Documentation, Documentation, and Documentation.
- Due Diligence.
- Responsibility and Accountability Within the Signatures.
- Independent Verification and Validation (IV&V).
- Using the Right Tools.

The roof is supplied by Air Force top-level support and management support. Many times, our management has stepped in to provide visible encouragement and support. To those in the field, this support is key to alleviating their concerns over the great amount of extra but necessary work this effort creates. Our corporate approach to Y2K provides the glue that holds it all together. We encourage and expect organizations to go beyond the continuing higher-level guidance and to make it work at their locations. Much of the guidance is in the process of being created, which demands creativity from our organizations. The guidance can be tailored, and suggestions for such are provided within each step, allowing precious flexibility for organizations to choose the way to make certification work for them.

Consistency Within the Process

This tenet starts with the selection of your certifier. The certifier is the most important person within the process and is therefore located at the center of this activity. The relationship between the certifier and the other roles within the certification process is displayed in Figure 1. Certifiers need to be carefully selected to be able to fulfill the many demands placed upon them. The section "Certifier Qualifications" provides guidance to help select certifiers.

Organizations are also encouraged to remove all conflicts of interest that can be realized, e.g., if the individual's job responsibility is directly within the same chain of command as the system. The systems programmer, system owner, system tester, and point of contact are all examples of individuals who have stakes in the system and therefore represent conflicts of interest.

A standard set of tasks has been developed for certifiers for each of the five phases in which they will be involved. This enables the same process to be followed each time with predictable results. Standard documentation has also been identified and issued with templates or guidance to be used to simplify the certifier's job. Often, document content varies. By making documents standard, certifiers can maintain more control of the outcome when interacting with the others. Standard roles and relationships for individuals involved in the process create an accommodating atmosphere.

Next, a standard, required training process is used to train the certifiers so they will be aware of documentation, tools, and how to implement them consistently. Finally, if the certifier meets the requirements and is trained, the process used to certify systems should not vary across the Air Force. Therefore, consistency in the process means getting the right type of person as your certifier, providing standard training to increase the consistency across all certifiers, and performing checks to verify the process.

Working as a Team

This tenet is based on the five standard roles and relationships illustrated in Figure 1 and defined below. As specified in the previous tenet, these important roles place certifiers in the center, but even though certifiers have undoubtedly the most important role due to the collection of various experiences and knowledge they must have, they are only one part of the whole. They need to build relationships with the other players to be successful. Following are the titles of the other individuals as shown in Figure 1 and their relationship with the certifier.

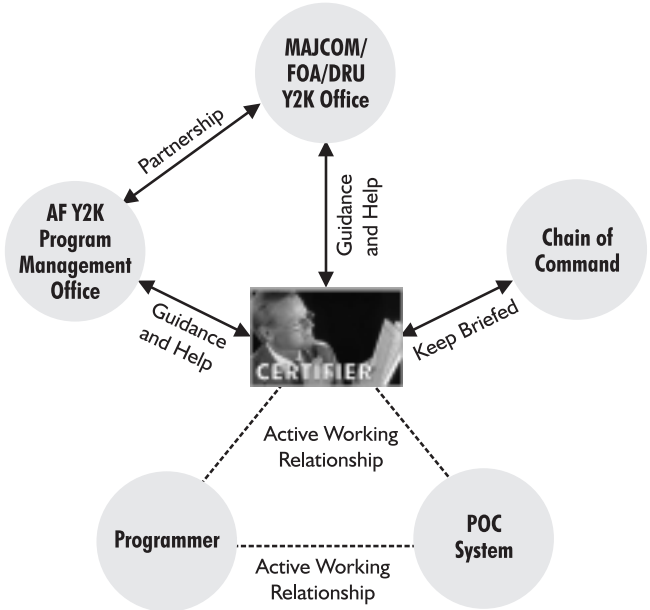
Year 2000 Program Management Office (Y2KPMO)

The relationship between the Y2KPMO and the certifier is one of support and reporting within already established major command (MAJCOM) Y2KPMO guidelines. It is never the intent to circumvent the established link between the certifier and the MAJCOM Y2KPMO. The Y2KPMO-certifier relationship should merely be an enabler toward certification. Since the Y2KPMO is responsible to assign certification control numbers (CCN) for systems, certifiers need to understand the reporting requirements to obtain them. The CCN is the control that identifies that the standard certification process has been applied to a system.

The MAJCOM PMO

This is the policy-making part of the organization. This group will determine the purview of all documentation, how the systems will be certified, who the certifiers are, what systems they are responsible for, and the extent of the Program Management Plan (PMP) for the organization (one overall broad PMP or several PMPs that represent separate business segments). The PMP must determine if all mission-critical and mission-essential systems receive independent test-type certification.

Figure 1. *Roles and responsibilities. The certifier is the "center" of certification activities.*



The Chain of Command

The certifier may be required to report to two chains of command. There will be the normal chain that involves the certifier's supervisor and the new addition of the systems chain of command. This relationship needs to be defined early to determine the needs of each party. Certifiers must feel confident with this chain of command because there may be times when they need their command's immediate attention to address a potential Y2K failure or symptom. The chain of command will need confidence that the certifier will only bring items tantamount to the success of the system to their attention. Therefore, this relationship is one of reporting and action. Care must be given to avoid unnecessary reporting to the chain of command, which can bog down the certification process.

The Programmer or Point of Contact (POC)

The programmer or POC is a generic representation of the person responsible for system development. In smaller organizations, the certifier may deal directly with the programmer. In larger organizations, it may be the system manager or POC. Depending on the software background of the certifier, this relationship may include formal and informal meetings. If certifiers have little software background, they may need to attend early meetings to determine responsibilities and types of reporting. Some of the other activities within this relationship are Y2K monitoring, assembling documentation (anticipating tenets four and five), and creating mutual Certification Tracking Document and Certification Level agreements. The more that is agreed upon in the Certification Tracking Document, the less that will be in contention later.

People in all roles need to work together as a team. If a single relationship is out of step, the potential for failure increases exponentially. For example, if certifiers find that actions taken by the development team are not consistent with the original plan, they must immediately bring this to the attention of the chain of command. It takes considerably longer to apply a fix once a system is in validation and has had a fixed window fix applied to it when the original strategy was for a four-digit year data fix. The same situation occurs if the chain of command does not speedily react. Fixes, if accomplished when discovered in the earlier phases, can be more easily applied.

Documentation, Documentation, and Documentation

Most of the documentation in question already exists in most organizations; therefore, providing documentation becomes as simple as creating the Y2K connections and additions to each piece. Just as documentation adds consistency to the process, the lack of documentation can cause disorder and increase the risk of failure and lead to interfaces that need rework or do not have anything to do with each other.

Taking the time to establish documentation early in the process will save time in the long run and reduce errors. Care must be taken to gather and organize the proper written communication that will maintain proper tracking through-

out the systems development or renovation. Along with providing a record of what was done, proper documentation helps all involved in the project better understand their roles and their relationship to others on the team. Maintaining documentation will not reduce all errors; however, when there are errors, the documentation will make it easier to track them down. The section "A Review of Required Documentation" lists all the required documentation and gives example content.

Due Diligence

Due Diligence will probably be the most important tenet after the dust settles. If your mission-critical or mission-essential system should fail due to a Y2K problem, you may find yourself testifying in court. Without evidence of due diligence, those involved could be held liable for the damage created by system failures—this is the case in the Air Force because the certifier must sign documentation to certify the system is Y2K compliant.

Recent news articles and comments by Air Force management have increased concern in this area, most notably through questions received during Air Force Certification training. Further, no one is "bullet proof," and the facts will determine how a case will be handled. In most instances, however, people who have been adequately trained, act responsibly, are competent in their job, put forth their best effort, and identify shortfalls that are beyond their control are at little risk. On the other hand, people who are incompetent, dishonest, or deceitful are at a higher level of risk. In the final analysis, there may always be some level of tension when managers try to allocate scarce resources and action officers try to apply these insufficient resources to complete the required tasks with due diligence.

We have defined due diligence as the necessary and earnest effort to accomplish a task. What does this mean? The definition is relative to the task being performed. Let us take it from the certifiers' perspective. First, certifiers must determine how their background fits in with the developers and the testers with whom they will be working. Some certifiers may not have the background necessary to understand everything a developer or a tester does. In this case, we recommend that during Air Force Certification training, certifiers accompany the developers and the testers to their routine meetings to better understand these jobs. We also recommend that the certifier and the developer together complete the Compliance Checklist and determine the level of compliance the organization has chosen for certification of each system.

Additionally, there are seven forms of documentation required for certification: Certification Tracking Document (CTD), PMP, Contingency Plan, Memorandum of Agreement/Interface Control Documents (MOA/ICD), Test Plan, Configuration Management Plan (CMP), and Program Implementation Plan (PIP). Because they are an integral part of the IV&V portion of the certification process, these forms must be developed for each system, and the certifier needs to not only be aware of them but also understand how each is set up and how they are integrated.

The CTD is the most important tool the certifier has, and if necessary, it is designed to be his best defense in a future courtroom. As the title suggests, the CTD is the official tracker for the system. When a Y2K problem occurs, if the CTD is completely filled out, the document should point to the area where the problem occurred. The CTD helps expose areas of risk, uncover areas left undone, identify potential problems, and identify what is left to be done.

The concur and nonconcur blocks are the most important part of the CTD. Each block allows the certifier to raise a red flag if things are not proceeding as expected or desired. Additionally, and often overlooked, the remarks section of each block is a large area reserved for the certifier's comments about what is being done. Although there may not be enough evidence to nonconcur, certifiers have the opportunity to express any risks that may be evident that could lead to failure. Therefore, filling out the remarks section with pertinent information becomes due diligence. It shows that certifiers are doing their jobs as the key players in the process. Without this information, it is suspect whether certifiers were involved in the process.

Some certifiers have allowed others to fill out the CTD for them, reasoning that the developer does the work and therefore can better answer the questions. This action or inaction can, in the worst case, lead to conflicts of interest—instances where the proper amount of time was not allotted to complete the document and other undesirable effects when the action of documentation gets in the way of development (from the certifiers' or programmers' perspective).

We included the job of the certifier in the process precisely to avoid this type of conflict. The certifier provides an objective look at the process. Developers and testers are too close to their work to effectively perform the certifier responsibilities contained in the CTD. Therefore, if this job is not done by the certifier—even though it does not lead to Y2K failure—it can be evidence that due diligence was *not* performed.

Since evidence can be used to defend oneself in court, it is important that the certifier maintain a folder for immediate and future perusal. This folder should contain copies of all required documentation as described above and any documentation deemed important to that system. For example, an IV&V performed by the Y2K PMO, audits by Air Force or DoD, and action plans to resolve deficiencies exposed by audits.

Responsibility and Accountability Within the Signatures

I have already discussed the importance of the CTD and what makes it an important document. However, all the information and data is meaningless unless accompanied by signatures of people in key roles. These signatures represent confidence in the work done and knowledge that everything possible was done to make the system Y2K compliant. Therefore, the responsibility goes back to all who were involved:

- The programmer or POC, lead developer, and systems manager (or however they are designated) will sign the CTD for every developer who worked on the system. They must know that their efforts and work were

complete and the total combination of effort will be Y2K compliant. Their signatures indicate that all the technical functioning and documentation is correct and complete. They must rely on most of the prior tenets to ensure their organizations are ready to document compliance.

- Certifiers are our center of activity; our coordinators, our unbiased view. Their signature attests that all bases were touched, everything went according to existing plans, and everyone who had a part played the part. The certifiers' eyes link all the players in the coordinated pattern represented by the documentation gathered.
- Users know what the system is supposed to do for them and the way the information should look. Their responsibility is to withhold their signatures unless their system is also proven to work to their specifications. This means that the documentation that is important to them (MOAs/ICDs) will be taken care of near the beginning to ensure direct compliance. Users are responsible for part of the certification work. It is essential that they are involved throughout the process to make sure things proceed along as they specified.

Systems accountability is provided by the signatures. If the system fails, the certifier, developer, and the user will share the fault and must work the fix together. Accountability works as a tool to enforce due diligence. The threat of the courtroom keeps all parties interested in the final outcome and ensures they do what it takes to get to Y2K compatibility.

Follow-Up Is the Key to IV&V

IV&V is the sixth tenet to certification and has been developed as a two-stage process. The first stage ensures that all tools gathered from the Certification training are in place, the Air Force Y2K Database (or Air Force Automated System Inventory) is cleaned up, and that further support is provided to the organization to get the certification process properly started. Discussions are initiated to develop the Certification Plan, the formal listing of certifiers and the systems they will certify, and to employ the proper initial documentation including the CTD. In this vein, follow-up provides a secure foothold that the original training initiated.

The second stage of IV&V is the actual verification and validation that certification has properly been done. In this stage, the database is analyzed for correctness, the systems are analyzed, and certifiers are interviewed to confirm consistent and standard application of the certification process. The organization then performs follow-up by taking all the findings from the second stage IV&V and developing an action plan to resolve any deficiencies or problems.

Using the Right Tools

Using the Right Tools and using them correctly is led by the CTD. All the required documentation is in the section "A Review of Required Documentation." There is no doubt these are the right documentation tools. The question is, "Were they used correctly?" Following are several of the tools and potential questions that can be asked.

CTD

- Did the certifier fill out all the blocks and initial all the activities?
- Did the certifier get management involved when the certifier was forced to nonconcur with an action?
- Were all nonconcurs resolved?

The Compliance Checklist

- Was the certifier present when the Compliance Checklist was completed?
- Did the developer complete the checklist during testing or after the tests were performed?
- Were there any comments as part of the checklist?

Contingency Plan

- Is a plan in place for all cases in which one is required?
- Are all Y2K considerations covered?
- Are system and operational items covered?

Test Plan

- Does the test plan include adequate Y2K testing?
- Does the regression testing include all the Y2K changes?

Certifier Qualifications

This section describes the characteristics and experience of the "perfect" certifier. We realize that individuals who fit this description realistically do not exist or are extremely hard to find. To find the best certifier for your systems, it is in your best interest to find a close match to the perfect certifier description. A certifier is designated by the commander and given proper authority and responsibility to meet the organization's and the Air Force's Y2K objectives. Selection criteria may differ depending on the functional area, mission requirements, and other circumstances determined by the commander. This document is a guideline that may be used by commanders to select a certifier.

Organizational Knowledge and Experience

- Knowledgeable of all key players, internal organizational, external functional, and command elements involved with systems of interest.
- Sufficient knowledge, skill level, and ability in the functional area of interest to allow effective and timely assessments and evaluations.
- Familiar with and displays an understanding of the complexities of the mission environment, systems, and applications they must certify.

Technical Knowledge and Experience

- Possess the technical and operational expertise to extract pertinent information during the certification process.
- Possess the ability to conduct a methodical and extensive lifecycle analysis of data, information, procedures, and processes related to the functional area assigned.
- To meet the technical criteria, a certifier should have some background in
  - the Capability Maturity Model for Software.

- quality assurance or IV&V.
- testing (planning or execution).
- software engineering.

Authority and Commitment

- Must serve in a level or position commensurate with the responsibility, possess the necessary authority to execute required actions, and have access to relevant databases and cognizant command authorities.
- Primary or "lead" certifiers must have sufficient time remaining in the assignment and position to perform the duties required until 2001 to ensure continuity.

A Review of Required Documentation

CTD

The CTD tracks the progress of certification for each system and reduces the risk of Y2K failure. This tracking provides management with greater assurance of success by the early identification and resolution of Y2K-related problems.

PMP

The PMP ensures that everyone, including people in your chain of command and any subordinate organizations, understands their roles and responsibilities in relation to solving the DoD and Air Force Y2K problem. The typical PMP contains

- information on the background of Y2K at the organization.
- a goal or purpose for the organization.
- direction or program strategy.
- objectives and management approach.
- responsibilities and resources.
- baseline milestones and tasks and schedule.
- annexes and support documentation.

Contingency Plan

The contingency plan maintains the continued success of the system by addressing every known or possible instance of failure and indicating alternative resolutions. Contingency plans address all possible known future occurrences of mission interruption—whole or partial. The plan identifies interim and permanent remedies and associated implementation timelines. Contingency plans also inform system users of possible work-arounds. Air Force Contingency Plans consist of system and of operational contingency plans. The System Contingency Plan is developed by the programming system manager, and the Operational Contingency Plan is formulated at the unit level.

Test Plan

The test plan establishes standard and consistent plans and scenarios for all Y2K testing for the development team to follow. Test reports document the results of the testing. The test plan typically includes

- scope.
- system and interface identification.
- referenced documents.
- test environment.
- software transition.
- test site.
- test items.
- personnel.
- test levels and classes.
- test conditions.
- test schedules.

CMP

The CMP establishes and maintains integrity of automated information systems (AIS) throughout the AIS lifecycle. The CMP institutes specific procedures to manage changes to AISs. The configuration of the AIS is analyzed at given points in time. Configuration changes are systematically controlled, and there is traceability of the configuration at each step.

MOA/ICD

The MOA/ICD provides complete agreement between all interface users of a system. Many ways to solve the Y2K problem have been identified. Some of these solutions do not work well with each other. The MOA/ICD forces communication and agreement on the chosen Y2K resolution method. This is handled during the assessment phase in the renovation strategy. In addition, costs for translators, etc., must be addressed and agreed upon within the MOA/ICD.

PIP

The PIP documents when, how, and who will field the Y2K-compliant system. A typical PIP contains

- introduction.
- purpose and scope.
- authority references.
- action agencies.
- actions required.
- milestones.
- resourcing.
- command and control.

In addition, appendixes usually cover

- time-phased actions.
- contingency management.
- automated information systems.
- sources of information.
- points of contact.

Conclusion

Certification adds to the confidence of Y2K compliance by measuring the process and adding the assurance of unbiased checking. This assurance is accomplished throughout the five-phased process in a checklist, or ordered, fashion. Organizations must start now before it is too late. The seven basic tenets are keys that can make previous guidance more understandable and attainable and serve as a measuring stick and a helping hand. Remember, our corporate approach to Y2K is the glue that keeps everything together and allows flexibility. This approach allows organizations to work where they feel more comfortable. The Air Force Certification process is designed to bring management's expectations in line with Y2K compatibility assurance. ◆

About the Author

Thomas V. Ashton is a software specialist and one of the pioneers of the Air Force Y2K problem resolution. In the Program Management Office, he leads the certification effort and was one of the key developers of the Certification Training Program, Certification Tracking Document, and Independent Verification and Validation Program. He co-wrote the "Year 2000 Air Force Challenges" and presented it at several conferences in 1996

See Y2K on page 29

Pro-Active Metrics

George H. Wedberg
McDonald Bradley, Inc.

Metrics can be used for more than measuring things. They can be viewed as the starting point for process improvement on a project and as a mechanism to improve communication and teamwork. This article summarizes what was learned in this regard over the course of several years as the metrics program was developed and managed for the U.S. Army Standard Installation/Division Personnel System – Version 3 (SIDPERS-3) project.

Project management metrics typically are used to measure the progress of a project and the quality of its output. They also may be used to monitor key parameters of the development process—for example, the stability of the requirements or the effectiveness of the technical review process. The principal benefit of a metrics program is improved *control* of the project; metrics furnish an overview of progress against plan, provide early warning of problem situations, and enable management to take corrective action.

That control can be significantly enhanced by taking a more pro-active approach to the metrics program—an approach in which the information obtained while gathering metrics data is used to improve the processes used by the project and the effectiveness of the project team. The metrics team is in a good position to observe the lower-level workings of the project, including the problems and inefficiencies that are hindering the progress of the developers and whether they are being addressed effectively. Information of this kind is an invaluable source of ideas for process improvement. It is also a starting point for building the management-developer bonds needed for a successful project.

The pro-active approach to metrics also focuses on communication. The success of a project depends, to a significant degree, on how the participants feel about the project, each other, and their management. People need to feel that they are accomplishing something of significance, that they can rely on each other and their management, that they are kept informed about project issues, and that their concerns are heard. Much of this can be accomplished by relatively simple communication mechanisms—particularly if that communication is backed up by action.

A successful project usually is the result of marrying technology and psychology. Both are necessary; neither is sufficient by itself. Technology usually prevails; we devote enormous energy to selecting the optimum software tools and hardware platforms. But we cannot forget that people want to work in a positive environment where they can make a solid contribution, exercise their creativity, and develop their skills.

During the course of the SIDPERS-3 project, we gradually evolved a pro-active approach that involved five specific actions designed to help create that type of positive environment on a team that peaked at 145 people. These actions were designed to involve the developers in the metrics process, to ensure they were heard with regard to process improvements, to improve communication at all levels, and to strengthen teamwork on the project.

Metrics Criteria

We selected the metrics to be used on the project and the way in which they would be used with an eye toward the climate we were trying to create. We understood that no one likes to be measured and that metrics are threatening enough as it is, so the intent was to not make it any worse. People who feel threatened pull into their shells and become defensive; they tend to tell you only what you want to hear. We wanted a climate in which they came out of their shells and told us everything.

We ensured that the metrics used also provided information that was useful to the development teams and that it was provided in a timely enough fashion that they could track their own progress. The metrics process was set up so that most of the data came from the developers; this helped ensure that the biweekly metrics reviews focused on progress and problems rather than on disputing the data. And finally, we made the metrics as nonpunitive as possible. The management team did not use them to judge people; metrics were not used as the basis for blame, threats, or performance reviews. Metrics were not used to compare teams; some of this is inevitable, but management did not encourage it.

The metrics team (consisting of one full-time person and two others part time) served as the proponent for this approach to metrics. We articulated the overall intent, convinced management to adopt it, and successfully argued the case, for example, with those who initially saw metrics as a convenient way to measure people rather than project accomplishments.

Biweekly Metrics Review

Every two weeks, the senior management met with the eight to 10 development team leaders and the metrics team to review the project metrics. Over time, we moved these meetings away from the initial inquisition-excuses-blame mode to a more positive mode in which we focused largely on removing the impediments that kept one team or another from meeting its planned objectives.

We changed the meetings by both convincing management of the advantages of a solution-based approach and taking a pro-active approach to identifying and removing impediments to progress. The problems faced by the teams were real: hardware that arrived late, repeated compiler problems, interteam process problems that necessitated excessive rework, and many more. Before each meeting, we asked

the team leaders to identify their current problems in writing. We addressed each problem in the meeting, developed a plan for dealing with it, and followed up after the meeting to ensure the plans were being carried out. The senior managers agreed to help effect timely resolution of the problems and left many review meetings with a list of calls to make.

In time, the development team leaders realized that putting a problem on the table would result in help rather than blame, and this encouraged them to surface problems rather than hide them. Giving the project manager information about real and potential problems strengthened his control of the project in at least two ways. First, he could deal with real problems while they were still small and relatively easy to resolve. Second, he could investigate potential problems and take mitigating action long before they affected the project. His control also was strengthened because team leaders and senior management were now working as a team with a corresponding increase in mutual trust. That the latter was not easily quantified made it no less real.

Implicit in this teamwork was the unspoken understanding that once the impediments to your progress have been eliminated, there is no longer any reason you should not make your dates.

Technical Reviews

Technical reviews in one form or another, e.g., walk-throughs and inspections, have long been acknowledged as technically useful. They are used to find problems early and to help prevent future problems. Don O'Neill described a software inspection program and its benefits in "Setting Up a Software Inspection Program." [1]

As one response to the number of software defects showing up in the metrics, the metrics team encouraged the development of a formal code inspection process by the software developers. The resulting process used extensive checklists, assigned roles and responsibilities for the review, and tracked the numbers and types of defects found.

The code inspections also increased the sense of teamwork on the project. Almost any well-run technical review process accomplishes this for several specific reasons. First, the process teaches people how to depend on one another, which is a key element of teamwork. They learn that accepting detailed evaluation of their material by their peers leads to a better product. They experience the satisfaction of being heroes when it counts most—at product delivery time.

The barrier that has to be overcome is ego. Few enjoy having their mistakes and bad assumptions pointed out, particularly in a group meeting. A good technical review process minimizes this problem by decriminalizing errors [2]. Errors are treated as a fact of project life; everyone makes them, no one is blamed for them. The idea is to make them visible, classify them, and learn from them. In time, people learn to react to their errors objectively rather than defensively, and a great battle has been won. The importance of subjugating ego cannot be overemphasized. A 1996 article in *Fortune* magazine described a number of world-class teams—from the U.S. Navy Seals to the Tokyo String Quartet. A common theme throughout the article is the lack of individual egos on these teams; every member is totally focused on the mission of the team [3].

The technical review process also builds teamwork because participants learn a new and useful process together. They learn how to evaluate a product against written requirements rather than personal preference. Experience with inadequate and incomplete requirements also leads to enlightenment about developing good requirements. Participants learn how to classify errors according to their source, which is a basis for preventing future errors. Participants also learn how to run a disciplined meeting, e.g., the purpose of the review is to identify errors, not resolve them; resolution of errors is the privilege and responsibility of the producer of the material [4]. In time, the participants learn that they can be more successful together than as individuals.

Empty rhetoric about "teamwork" is widespread in the business community; technical reviews are one concrete way to implement the rhetoric.

Communication

The goal of our communication processes was to establish as open an environment as possible—one in which people were comfortable surfacing problems and telling us what they needed to be more effective. We used several mechanisms in addition to a lot of one-on-one discussion:

- We conducted an informal written survey of all members of the project that asked them what they thought was going well and not so well, what problems they were having, what they would like to see changed, and the like. The objective was not measurement but rather to see what they had to say. Although the survey was anonymous, we invited those who were concerned about having their E-mail traced back to them to respond any way they chose, including notes under my door at night. Because a few did the latter indicates the difficulty of establishing trusting communication.
- We conducted skip-level meetings in which programmers, testers, etc., met in groups of a dozen or so with the project manager—no other managers or team leaders were present. The participants were encouraged to bring questions from their teammates as well. This was an opportunity to discuss rumors about high-level topics such as project direction and funding, to surface frustrations directly to senior management, and to understand and influence the project manager's thinking. This direct contact, somewhat unusual on a project of this size, let the project manager explain his priorities and reasoning without the usual middle management filters and gain a direct understanding of what was important to the employees he supervised. Out of these meetings also came a number of "social" changes to the project, e.g., the occasional holiday party evolved

into monthly project luncheons, and casual Friday became casual summer, then became permanent casual.
- We conducted lessons-learned exercises after each major increment of the project. The metrics group collected inputs from all project areas, organized them by subject area, and facilitated working meetings to discuss problems, working relationships, and potential process improvements. The documented outputs of the meetings were the basis for process improvement activities.

The communication mechanisms, in addition to revealing problems, also raised the level of trust on the project. As might be expected, the developers sometimes used their communication opportunities to challenge management, in effect, to improve some aspect of the project. When management responded promptly with meaningful actions, both sides knew that everyone's level of commitment had just risen.

Process Improvements

The biweekly metrics reviews, the lessons-learned exercises, and the other communication mechanisms brought out problems and frustrations of all kinds—from inadequate technical planning to lack of vendor support to poor working relationships between project teams. All such concerns were analyzed to determine their underlying causes. More often than not, the cause was a poor or nonexistent work process; this was particularly true of problems involving friction between project teams.

Of course, the configuration management (CM) team was unhappy that developers were turning over code to them before it was properly integrated; investigation showed there was no well-defined process for integration and turn-over. And the coders were annoyed at having to rewrite modules numerous times because the database kept changing; investigation showed there was no visible plan or schedule for database changes.

Some of the resulting process improvements were accomplished simply, e.g., a verbal agreement between two individuals to exchange key information at regular intervals. Other improvements required analysis, documentation, and review. The CM problem noted above led to a diagram showing the sequence of all steps required to manage software modules from the developer's unit test onward; this included the definition of each step, the person responsible for accomplishing it, the machine on which it was to be performed, and so forth. Project-wide distribution of that diagram significantly reduced the friction between the developers and CM while increasing project productivity. This, like most process improvements, strengthened the project manager's control of the project by making the software development process more predictable.

Conclusion

The pro-active approach to project management metrics—a combination of measurement, process improvement, and communication—strengthened the project manager's control of the project in several ways.

- The measurements provided an objective picture of the project's progress, status, problems, successes, and failures. This factual information provided the basis for subsequent management and technical decisions.
- Process improvements made many aspects of the project more predictable. Every process improvement that standardized a procedure or eliminated an impediment, for example, made it more likely we would get the desired result in the expected time frame.
- The open communication environment meant that we got useful input from a wide variety of people at all levels of the project. We also believe, but cannot prove objectively, that there were many intangible benefits from that environment—people who are both informed and heard, and thereby involved, have a higher level of commitment. We saw that commitment on many occasions.

Each of these practices emphasizes the role and needs of the project's individual contributors and thereby strengthens their connection to the project and its success. It is difficult to remain disinterested or cynical when the work is going well and the project is helping you meet your personal goals. ◆

About the Author

George H. Wedberg is a program director with McDonald Bradley, Inc. He has over 20 years experience with all aspects of the software development lifecycle, in both the federal and the commercial sectors. Past accomplishments include development of the first software engineering program for General Electric Information Services Company, development of the metrics and risk management programs for the SIDPERS-3 project, and creation of measures of effectiveness for the Department of Health and Human Services and for the U.S. Marine Corps. He holds a doctorate in physics from Indiana University.

McDonald Bradley, Inc.
Suite 805
8200 Greensboro Drive
McLean, VA 22102
Voice: 703-827-9376
Fax: 703-827-8604
E-mail: wedbergg@erols.com

References

1. O'Neill, Don, "Setting Up a Software Inspection Program," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, February 1997.
2. Rifkin, Stan and Charles Cox, "Measurement in Practice," *Carnegie Mellon University Software Engineering Institute Technical Report CMU/SEI-91-TR-16, ESD-TR-91-16*, July 1991.
3. *Fortune*, Feb. 19, 1996, pp. 90-99.
4. Freedman, D.P. and G.M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, Dorset House, New York, 1990.

Are *You* Ready to Deliver? To Ship? To Test?

Capt. Brian Hermann, *U.S. Air Force*
Jim Russell, *Honeywell, Inc.*

How do you know when you are ready to deliver your software product? Do you ship on your contract delivery date? Are you under pressure to get the product to market and beat the competition? What kind of measures and metrics do you use to make the decision to deliver? This article introduces the method the Air Force Operational Test and Evaluation Center uses to determine whether a system's software is mature enough for the system to enter dedicated operational test and evaluation and to subsequently be fielded or procured. Though we use this evaluation to determine test readiness and suitability for fielding and procurement, it could easily be applied as an exit criteria for software development or as a component of the "decision to ship" process.

Software product maturity: *a measure of the progress software products are making toward satisfying user requirements* [1].

The Air Force Operational Test and Evaluation Center (AFOTEC) uses a software product maturity evaluation as a test readiness criterion for a system's software prior to a system's entry into dedicated operational test and evaluation (OT&E). Our evaluation is a trend analysis of software changes identified during software development and testing. Based upon this analysis, AFOTEC provides a recommendation of the software's readiness for operational testing. This evaluation currently does not provide any kind of projection of future software product maturity but provides an excellent snapshot of system maturity. (However, AFOTEC has performed some investigation into methods of maturity projection [2].) The entire method is documented in [1].

Software Product Maturity Data Requirements

Our experience shows that most developers and procurement offices already collect the data necessary to perform a software product maturity evaluation. Data that describes and tracks documented software changes serves as the key input to the evaluation. Following are the minimum data required for each software change to evaluate software product maturity.

- Software change (problem) number.
- Description.
- Computer Software Configuration Item (CSCI) Identifier.
- Severity level.
- Date change opened (or problem found).
- Date change (problem) closed and implemented.

For the change Severity Level definitions, AFOTEC adapted the "Priority Classifications for Problem Reporting," listed in Appendix C of MIL-STD-498. Using these definitions, systems with open Severity Level 1 or 2 software changes are not recommended for entry into dedicated OT&E. Many organizations use different severity, criticality, or priority definitions. Any reasonable ranking system is acceptable as long as a clear definition of product maturity is included.

Software Changes

By software *change*, we mean any change that

- Corrects errors (corrective change).
- Enhances system capability (perfective change).
- Makes the software compatible with changes in the computing environment (adaptive change).

In our evaluations, we include software problem reports, software failure reports, software change requests, trouble reports, and any other data that fits the above definitions. If the software does not meet user requirements, the documentation of the unmet need is an input for the software product maturity evaluation.

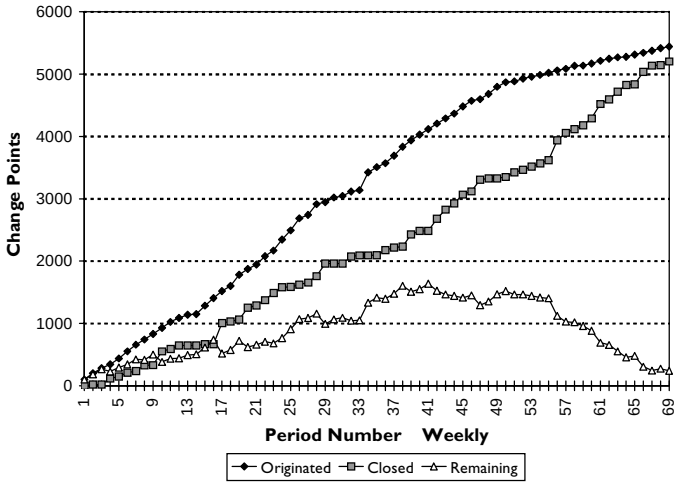
External Factors

To correctly gauge readiness to deliver, developers must also evaluate test completeness, test rates, and requirements stability. Any of these factors can cause product maturity to look unrealistically good or bad. Obviously, if only 10 percent of the planned tests have been completed, it is premature to ship the product—despite low software change trends. Likewise, high test rates will likely produce more changes and problems than lower test rates. Requirements instability is one of the most common causes of software product immaturity of the Department of Defense's long development cycle projects.

Maturity Evaluation and Analysis Tool

AFOTEC developed a Microsoft® Excel for Windows™-based tool, called Maturity Evaluation and Analysis Tool, to

Figure 1. *Accumulated software changes (weighted).*



automate the data manipulation, produce trend charts, and speed analysis and reporting. The tool and user's manual are available to U.S. government offices and their contractors at no cost from HQ AFOTEC's Software Analysis Division at DSN 246-5310 or E-mail sas@afotec.af.mil.

Trend Charts and Analysis

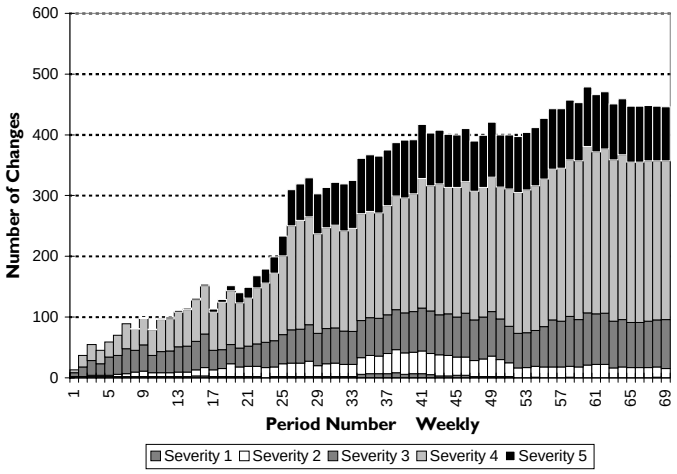
Software product maturity evaluation entails a graphical analysis of change data trends in the context of project schedule and other external factors. The basic product maturity chart (Figure 1) shows the *total changes originated*, *closed*, and *remaining* trends. (Note: These charts contain data from multiple, real systems and are provided as examples only.) To indicate maturity or progress toward maturity, the total changes originated trend should begin to level off. This indicates testing is finding problems at a decreasing rate. If problems are being closed efficiently, the total changes closed curve should closely follow the total originated trend. Ideally, all identified changes are closed, and the remaining changes curve would show no backlog.

Although the *remaining changes* trend in the basic chart shows the current software problem or change backlog, Figure 2 presents a more useful view. This stacked bar chart shows the overall backlog trend as well as each severity level's contribution to the total backlog.

Figure 3 shows both remaining changes for each CSCI and the *defect density* (the number of remaining changes or problems divided by thousands of new or modified source lines of code). In addition to the minimum change data, defect density analysis requires code size information. Literature suggests software is not ready for release until the defect density is below 0.5 [3]. Rather than blindly endorse this number, we suggest developers select a threshold of their own. Finding portions of software with the most remaining problems and the highest defect densities are two additional pieces to the product maturity puzzle.

Our product maturity tool produces over a dozen additional trend charts including average severity, severity level

Figure 2. *Remaining software problems (unweighted).*



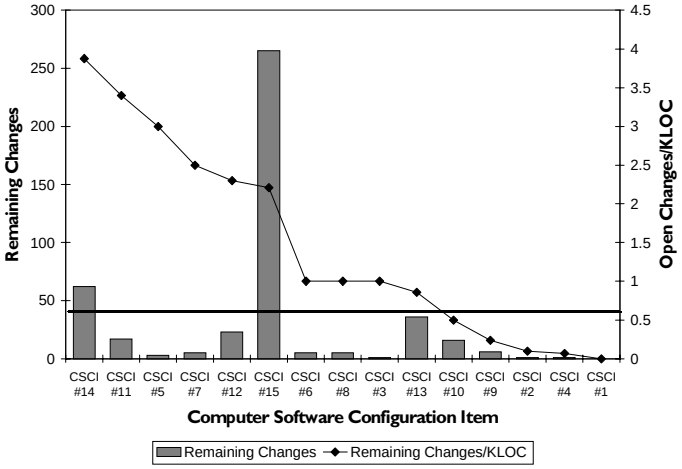


Figure 3. *Remaining changes and defect density.*

distribution, average closure time, charts for each severity level, and charts for each configuration item or subsystem.

Value to Software Developers and Software Professionals

The AFOTEC approach for evaluating software product maturity is directly transferable to any software development activity and has much more value to a software developer than to an operational tester. Table 1 (page 30) shows just a few possible uses of the software product maturity evaluation.

Conclusion

Some of you may be thinking "so what." Yes it is true—this data is usually collected and readily available. Beyond managing rework, however, few developers take full advantage of this data housed in their own configuration management systems. Developers, as owners of the data required to perform a software product maturity evaluation, are best able to perform this evaluation and use the results to improve the quality of their products. Having metrics to back up your answer to the "are you ready …" question will help base your decision on quantitative facts vs. reliance on your gut feelings.

As former software maintainers, we wish we had been able to see the snapshot this maturity evaluation provides. It has been said that a "picture is worth a thousand words." What is the picture of your software product's maturity worth to you? ◆

About the Authors

Capt. Brian Hermann, U.S. Air Force, is an Air Force Institute of Technology (AFIT) computer science doctoral student at Arizona State University in Tempe, Ariz. His current area of study is software engineering. He was previously assigned to the AFOTEC at Kirtland Air Force Base, N.M. He has a master's degree in software systems management from AFIT and a bachelor's degree in electrical engineering (computer concentration) from the University of Notre Dame.

See READY TO DELIVER, *page 30*

Using Statistical Process Control with Automatic Test Programs

David B. Putman
Technology and Industrial Support Directorate,
Software Engineering Division, Hill Air Force Base

Many program managers are struggling with trying to meet their requirements with a declining budget. Statistical Process Control (SPC) is a tool that may help program managers get a greater return on investment. This study used historical test information to determine whether SPC should be used as a normal part of developing and supporting automatic test programs. The findings suggest that significant cost savings, with increased quality, can be gained by using SPC techniques.

Statistical process control is a method that allows users to separate random variations in their data from nonrandom variations, then analyze the nonrandom variations to improve the quality and reduce the cost of products. Conservative calculations in this article suggest that program managers can save millions of dollars over the life of a weapon system by applying statistical process control theory to automatic test programs. In this article, I use historical test data to show how statistical theory can be used to improve test programs. It is intended for readers with some basic knowledge of statistics but does not get into mathematical derivations because commercial applications ranging from spreadsheets to statistical process control products can perform the statistical calculations.

In my experience, SPC has been an overlooked tool in supporting the test programs associated with Automatic Test Equipment (ATE). Because of my success with SPC techniques in the field of ATE and circuit cards, these are used in the examples. However, SPC techniques can be applied to many other areas in which the user can break the repeatable process down so that the statistics can be applied to well-defined, repeatable, and measurable steps.

Advantages of Using SPC Techniques

There are many advantages to using SPC techniques during the development and maintenance of test programs. SPC techniques identify the true performance capability of the test program in relationship to the circuit card's performance and the test station's ability to test the circuit card. The use of SPC techniques will help reduce the overall lifecycle cost associated with the test program, including

- Eliminating or reducing the possibility of performing repair actions on a good circuit card.
- Reducing the chance of changing tolerances that would allow a faulty circuit card to pass.
- Reducing the chance of sending a faulty circuit card to the supply system as a good asset.
- Eliminating or reducing the tweaking of tolerances and test procedures that occur over the life of a test program, usually one at a time. By contrast, all of the potential problems identified through these techniques could be addressed in a single software update.
- Ensuring invaluable information is available if the tested item ever becomes obsolete. If a circuit is redesigned, it would be logical to assume the design engineer would design the new circuit to perform at the center of the testing tolerances. There is a potential that the new circuit will not function properly in the system if the older circuit performed closer to the edge of the tolerance.

The following example shows how you can estimate your potential savings by requiring the use of SPC techniques. Data for your specific project would be needed to determine your potential savings.

- Assume, for instance, that on the average, the use of SPC identifies 10 *changes* that need to be made on the test programs (C = 10).
- Assume that you are managing 100 analog *test* programs (T = 100).
- Assume there is an average $10,000 *price* per software update (P = $10,000).

If the problems are identified and corrected one at a time, your cost is

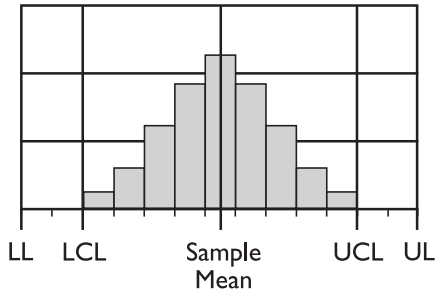
C*T*P or 10 * 100 * $10,000 = $10 million

If, by using SPC techniques, all problems are identified and corrected in a single update, the cost to the program manager is T*P = 100 * $10,000 = $1 million. This saves you $9 million in software updates. These calculations do not include the potential savings from reducing the rework costs.

Where to Start

To begin using statistics to control processes, you should identify what should be measured, the sampling method, and the data that exists. Statistics can be applied to both the product and the process. The product data includes conformance to specifications, whereas the process data includes the cost, schedule, and defect data. To use statistics on the process, the user must be able to break

Figure 1. *The desirable relationship is for control limits to be inside the test limits.*



the process down so that the statistics can be applied to well-defined, repeatable, and measurable steps.

The Sampling Method

It is often cost prohibitive to perform a manual measurement on every product produced. In this case, you need to address a method of using sample sets, such as randomly sampling 10 units every week. In other cases, automated systems may give you a 100 percent sampling capability.

Existing Data

In the case of process data, it is often difficult to determine exactly what should be measured, how to measure it, and how to display the data in a useable manner. When implementing the SPC concept, you may immediately develop new measurement schemes to capture "useable" data. Then, after taking process measurements over time, you may find that the data you are gathering is not giving you the information you want. Applying statistical theory to the existing data will help guide you when making changes to how the events are measured.

SPC Review

The run chart and histogram are two of the easiest methods to display historical data. These two charts provide much information about each of the tests performed on the circuit card. The histogram in Figure 1 shows the desirable relationship between the upper and the lower test limits (UL and LL) in comparison to the upper and the lower process control limits (UCL and LCL). The desirable relationship is for the control limits to be within the limits defined by the test program.

When the control limits fall outside of the test limits, as shown in Figures 2 through 4, many items will be reworked, thrown away, or salvaged for parts, resulting in higher production costs. The process may also incur additional intangible costs. An example of an intangible cost is customer dissatisfaction, which may result from paying excessive costs, lengthy delays, poor quality, or receiving faulty products. When the control limits fall outside of the test tolerances, as shown in Figures 2 through 4, the following questions should be asked.

- Can the manufacturing process or (in the case of ATE) the repair process be changed to reduce the rework costs and improve the quality?
- Are the specification limits correct?

Applying SPC Concepts to Our ATE Test Programs

ATE Test Tolerances

The derivation of the ATE test tolerances for electronic circuits can typically be traced to one of the following three methods.

- The parameter was specified in the original design requirement.
- A calculation was made to determine the theoretical performance of the circuit. This could be the result of a manual calculation or a computer circuit simulation.
- A measurement was made on a good circuit card, often called a "golden board," and then a +/-n% tolerance was added to the measurement.

Figure 2. *An undesirable condition – the lower control limit is less than the lower test limit.*

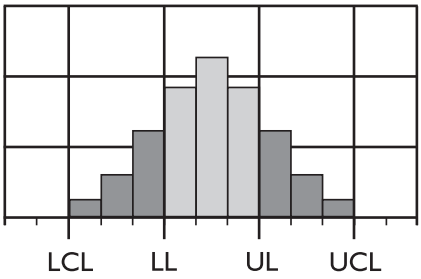


Figure 3. *An undesirable condition – the upper control limit is greater than the upper test limit.*

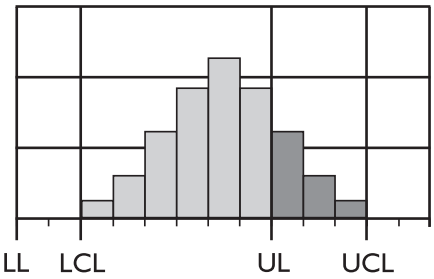
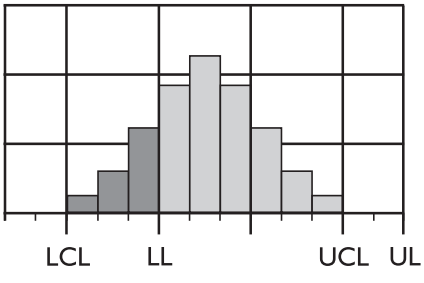


Figure 4. *An undesirable condition – both test limits are inside the control limits.*



From an SPC viewpoint, none of the methods discussed above will identify the capabilities of the circuit and the test station's ability to test the circuit. The verification of the system design is based upon a First Article Test, which usually verifies only that the system meets the design requirements. The number of inputs and outputs tested at the system level may be significantly less than the total number of input and output pins on all of the circuit cards internal to the system.

This problem is further complicated when the government test program is not hosted on the factory test equipment. The attributes of the ATE may be different on the depot test station than they were on the factory test equipment. The test station's attributes, such as input impedance, cross talk, and insertion loss, have a direct impact on the results of the test. Typically, the ATE attributes are not considered when the test specifications are developed for a circuit card.

Data and Assumptions

During this study, I was fortunate to be able to collect approximately two years of test results for a particular type of circuit board. Approximately 195 circuit cards of the same type were tested during this time. Surprisingly, our ATE does not provide us with an easy means to capture the test results and store the information into a format that can be easily used. Commercially available ATE may provide this capability, but I am unaware of any ATE specifically designed for use with a government weapon system that provides this capability.

The historical test results did not include the UUT serial number, which may have resulted in the same item appearing more than once. I assumed that once all of the tests passed, the station operator would not rerun the test program on the same UUT. I also lim-

Terminology

Common terms and acronyms used with statistics and ATE.

ATE**:** Automatic Test Equipment, which is used in this case to test electronic circuit cards.

UUT: Unit Under Test, e.g., the electronic circuit card being tested.

TPS: Test Program Set, which includes the software necessary to test the circuit card and the interface device between the circuit card and the test station.

LL: Lower limit defined in the test program. The LL may also be called the lower test limit.

UL: Upper limit defined in the test program. The UL may also be called the upper test limit.

TEST Number: This number, such as Test 791007, refers to the programmer-defined file statement number in the source code of the test program.

Mean (m): The average of the samples.

Standard Deviation (s): A way to show how the samples are distributed in relation to the sample mean. For a normal distribution, 68 percent of the samples are within m +/- 1s, 95 percent of the samples are within m +/- 2s, and 99.7 percent of all samples are within m +/- 3s.

LCL: Lower Control Limit, which for this study I set for the process at m - 3s.

UCL: Upper Control Limit, which for this study I set for the process at m + 3s. Statistically, 0.3 percent of the measurements will fall outside of the range I selected to calculate the LCL and UCL. Measurements outside of the control limits do not necessarily indicate that there is a problem; further analysis is required to determine whether there is a problem with the process.

Sample Set: Often, it is economically unfeasible to perform a 100 percent sampling of the products being produced. Statistics allow the user to perform a random sample, such as randomly testing 10 items at the end of every week. Through the use of statistics, the user can predict, with a reasonable certainty, the attributes of the unmeasured items based upon the results of the items that were measured. The X(bar) and R control charts are commonly used when the data is gathered in sample sets.

Histogram: The histogram is a way to graph the frequency (how often) the measurement occurs. The histograms in this article were graphed so that the LL corresponds to the left side of the histogram and the UL corresponds to the right side of the histogram.

Run Chart: A simple way to graph each measurement as it is made. Adding both the control limits and the test limits to the run chart provides an easy way to compare the information.

R Chart: A graph of the range (difference) between the largest and the smallest measurement in each sample set taken over time.

X(bar) Chart: A graph of the average of each sample set taken over time.

ited the data to only the test program executions that resulted in a UUT-passed message. This gave me an indication of the condition of the UUTs when they were returned to supply as serviceable. To perform a study of this nature, I recommend that the following information be collected for each test.

- Date and time.
- UUT part number and serial number.
- Test station serial number (if more than one may be used).
- Test number.

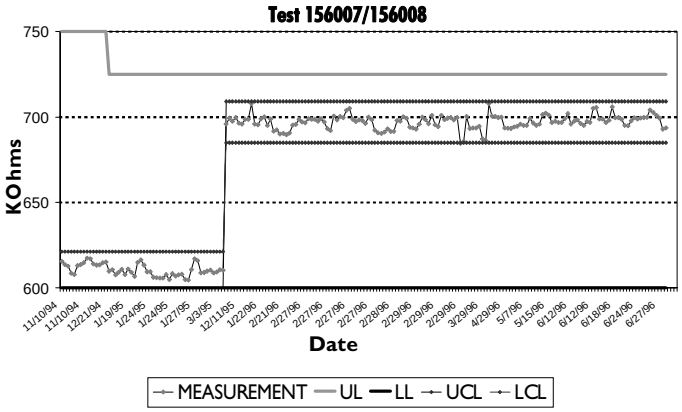


Figure 5. *The run chart can be used over time to show the results of the measurements and the effects of process changes.*

- Measurement and the units (such as ohms and VDC).
- Upper and lower test limits.
- Pass or fail information.

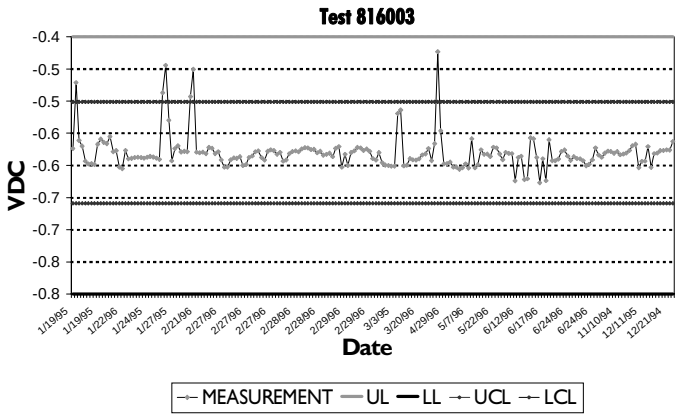
Process Changes

When calculating the upper and the lower control limits, it is important to take into account process changes. The run chart in Figure 5 shows that the UL was lowered in early 1995, and a second software change was made in early 1996. The first change had no effect on the measurements being taken or on the process control limits. The second software change had a significant impact on the measurements, which in turn shifted the process control limits. The measurements shifted from the lower end of the test limits toward the upper end of the test limits.

The first set of process control limits, UCL_1 and LCL_1, were calculated using only the measurements taken before the process change in 1996. The second set of process control limits, UCL_2 and LCL_2, were calculated using only the measurements taken after the process change in 1996. The process capability would appear to be much wider, and therefore worse, if the standard deviation was calculated using all the measurements in a single calculation.

Displaying the run chart as shown in Figure 5 allows for a quick visual comparison of the process capabilities in relation

Figure 6. *The run chart can be used to spot anomalies in the test data.*



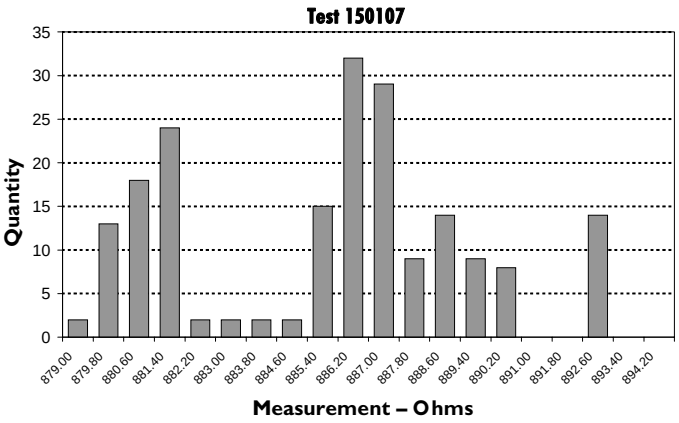
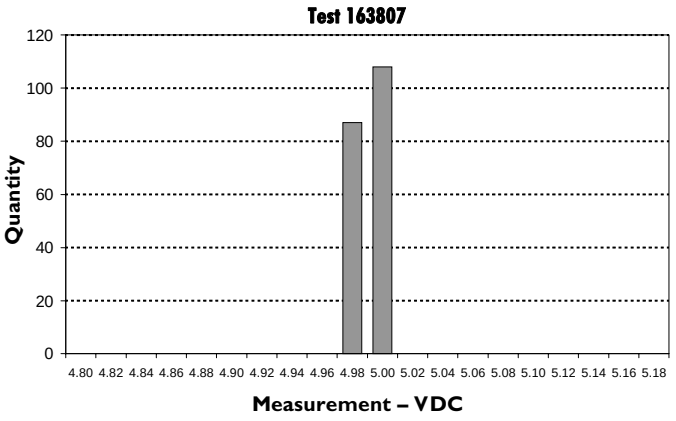


Figure 7. *The run chart for test 150107 shows three peaks in the measurements.*

to the test limits. This chart is a good example of how the use of SPC techniques may have driven a different outcome than the software changes associated with the two software releases. In the first software change, the programmer lowered the upper test limit. I was unable to locate the documentation associated with the change, but I assume the change was probably driven by a UUT that failed in the next higher assembly, yet passed at the upper margin of the test tolerances.

In a similar manner, I believe that the second change was made because good assets were falling slightly below the lower limit. The lower control limit at that time was lower than the lower test limit. In the second change, the programmer chose to add a delay to the program and change the scale on the multimeter. The result of the second change is the sample mean shifted from the lower edge of the tolerances toward the upper edge of the tolerances. The range between the upper control limit and the lower control limit was basically the same. I suspect that if this run chart had been available when the first software change was made in 1995, both the upper and the lower test limits would have been analyzed in more detail, and a different solution would have been implemented.

Figure 8. *This test measures the 5VDC applied to the circuit card. All of the measurements for the bar on the left were taken on station No. 5; all measurements for the bar on the right were taken on station No. 2. In this case, the minor calibration difference between the two stations does not negatively impact the test results.*



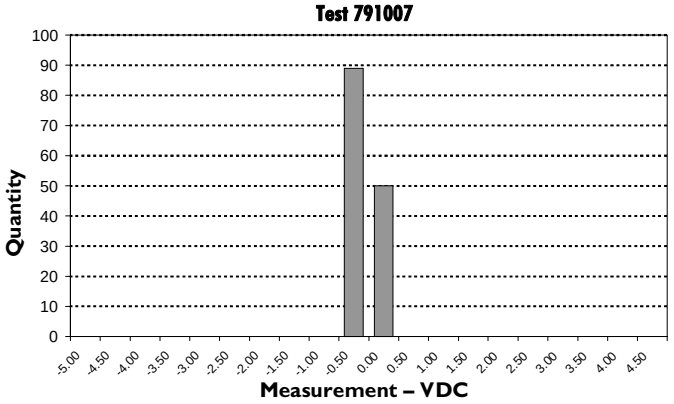


Figure 9. *Is this too good to be true? The process capability ratio (UL - LL) / (UCL - LCL) = 1250. Note: This equation is slightly different from the more familiar C_{PK} equation.*

The second software change in 1996 would not have been required.

Test 816003 (Figure 6) matches the desirable relationship discussed with Figure 1. Statistically, I expected 0.3 percent of the measurements to fall outside of the control limits. If the test limits are correct, I can be confident that the six circuit cards that measured between the upper control limit and the upper test limit will work properly in the system. The SPC user will be able to spot potential problems by watching these anomalies in the data. Problems with substitute parts, drifting calibrations, and aging or degrading components can be detected by looking for nonrandom anomalies and trends in the data.

Multiple Peaks in the Data

The use of multiple vendors or multiple measurement devices may cause peaks and valleys in the histogram, as shown in Figures 7 and 8. These peaks are another indicator that there may be a problem with the process. Identifying the causes of the peaks can be used to improve the procedures to procure parts, improve the procedures to calibrate the test equipment, etc.

Process Capability

The run chart and histogram can also show a cause for concern when the process capability is too good to be true, as is the case with Figure 9 for Test 791007. The tolerances for this test may be too wide; the program allows for a 10.000 VDC range between the lower and the upper test limit (-5.0 VDC to + 5.0 VDC). The data revealed that the range for the 139 circuit cards tested was only 0.0008 VDC between the lowest reading for the bar on the left (89 samples) and the highest reading for the bar on the right (50 samples). Knowing this information, I would suspect that I had a bad circuit card if it measured 4.50 VDC.

Results of the Study

Sixty-five tests for one UUT test program were analyzed in this study. The first three findings discussed below relate to Figures 2 through 4. Both the repair process and the testing tolerances should be reviewed for Findings 1 through 3 to determine whether the excessive rework could be reduced or eliminated. Reducing the rework effort results in lower costs to the customer. The findings included:

- In nine of the 65 tests, the LCL was less than the LL.
- In seven of the 65 tests, the UCL was greater than the UL.
- In three of the 65 tests, both the upper and the lower control limits were outside of the testing limits.
- Thirty-four of the 65 tests resulted in ratios of the test limits to the control limits (UL-LL) / (UCL-LCL) greater than five. This suggests that further analysis should be performed to determine if the tolerances are too wide and allowing faulty circuits to pass.
- Using the equation (Sample Mean – Center of test limits) / (3 standard deviations) revealed that for 13 of the 65 tests, the sample mean was shifted from the center of the tolerances by more than three standard deviations. This also suggests that further analysis should be performed to determine if the tolerances are correct. Genichi Taguchi and many other quality experts stress that the cost of quality rises if the process is not centered within the tolerances.

Conclusions

SPC techniques should be used as a tool to support test programs on automatic test equipment and similarly structured work, because conservative calculations suggest that millions of dollars could be saved by applying these techniques. Resistance to change or a reluctance to admit that the previous process was not perfect may hinder efforts of this nature. Program managers, shop operators, and engineering staff need to work together to assure that data is collected and used in an optimum manner. They also need to assure that future ATE procurement activities guarantee that the test results can be easily captured, manipulated, and displayed as shown in this article. ◆

About the Author

David B. Putman is lead of the Software Engineering Process Group in the Software Engineering Division at Hill Air Force Base, Utah. He has over 17 years experience in ATE, two years with Hughes Aircraft and over 15 years with the Air Force. He was the senior engineer within the Avionics Software Support Branch for nine years, and he supervised ATE engineering teams for two years. He also supervised the F-16 Operational Flight Program System Design and Integration Test teams for one year. He has a bachelor's degree in electrical engineering from the University of Utah and a master's degree in business administration from Utah State University.

OO-ALC/TIS-3
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: 801-777-4726
Fax: 801-775-3023
E-mail: putmand@software.hill.af.mil

References

1. Crow, Edwin L. et al., *Statistics Manual*, Dover Publications, Inc., New York, 1997.
2. Gujarati, Damodar, *Essential of Econometrics*, McGraw-Hill, Inc., 1992.
3. Chase, Richard and Nicholas Aquilano, *Production & Operations Management*, IRWIN, 1992.
4. Schmenner, Roger W., *Production/Operations Management*, Macmillian Publishing Company, 1990.

Metrics for Predicting Run-Time Failures and Maintenance Effort: Four Case Studies

Aaron B. Binkley and Stephen R. Schach
Vanderbilt University

It is important to have the ability to predict which modules in a software product are likely to be fault-prone so that corrective action can be taken. A number of metrics have been put forward to predict faults and failures, including the coupling dependency metric (CDM). In the four case studies presented here, CDM outperformed a wide variety of competing metrics. The case studies were implemented in COBOL, C, C++, and Java and used both classical and object-oriented methods. This article is a summary of research conclusions that can be examined in more detail by consulting the references at the end of this article.

The distribution of faults within any particular software product is not uniform. For example, 47 percent of the faults found by users of OS/370 were associated with only 4 percent of the modules [1]. Therefore, what is needed are metrics that can be applied to detailed designs to predict which modules will be fault-prone. Then, instead of wasting money detecting and correcting faults during implementation and integration, those modules can be redesigned before coding commences.

Metrics of this kind can also be applied to existing software, to predict which modules are likely to contain residual faults when the software is installed. Residual faults can have two major adverse consequences: run-time failures, which prevent users from making optimal use of the software; and time that has to be spent on corrective maintenance. If a fault is detected in an existing module that has been flagged as fault-prone, it may turn out cheaper to discard, redesign, and recode the module than to attempt to fix one fault in a module that probably contains many other faults.

The CDM is a metric that accomplishes this objective. It has outperformed a wide variety of other metrics, both classical and object-oriented, in predicting run-time failures and residual faults [2]. In this article, we compare the performance of CDM with other metrics on four real-world case studies: run-time failure data for a COBOL registration system, maintenance data for a C text-processing utility, maintenance data for a C++ patient collaborative care system, and maintenance data for a Java electronic file transfer facility.

Description of Method

We followed the same procedure in each of the four case studies. We chose a set of metrics that would presumably be good predictors of run-time failures or corrective maintenance for that case study. We then applied each metric in the set to each module in the case study. For example, one of the metrics was lines of code (LOC), so we counted the LOC in each module. We then chose an objective measure of software quality, for example, the number of faults detected in a module after installation. Next, we computed the Spearman rank correlation [2] between the LOC in each module and the number of faults in each module. A high correlation would indicate that modules with many LOC also have many faults, i.e., a high correlation would mean that lines of code is a good predictor of faults.

The Metrics in Our Case Studies

We compared a number of different metrics in each case study, including

This is a brief description of the metrics mentioned in this article. For detailed information on all the metrics in our four case studies, consult the Internet address listed in [3].

Lines of Code. We computed lines of code as the number of noncomment source statements.
Cyclomatic Complexity. The cyclomatic complexity metric measures the number of branches in a module.
Fan-in/fan-out. For any module M, the fan-in of M is the count of the modules that call M plus the number of global data elements. The fan-out of M is the count of modules called by M plus the number of global data elements altered by M.
Ordinal Scale Coupling. Level numbers are assigned to classical coupling categories. Specifically, level numbers 1 through 5 are assigned to data coupling, stamp coupling, control coupling, common coupling, and content coupling, respectively. The level numbers corresponding to each instance of coupling in a given module are then summed.
Response for a Class. The response set of a class is the set of methods that can potentially be executed in response to a message received by an object of that class. The response for a class (RFC) is then the number of methods in its response set.
Coupling Dependency Metric. The coupling dependency metric (CDM) is the sum of three components: a measure of the extent to which a program relies on its declarations remaining unchanged (referential dependency); a measure of the extent to which a program relies on its internal organization remaining unchanged (structural dependency); and a measure of the vulnerability of data elements in one module to change by other modules (data integrity dependency).

widely used quality metrics like cyclomatic complexity, object-oriented metrics like the depth of the inheritance tree (for the two object-oriented case studies), and CDM. The metrics compared in the four case studies are described in detail in [3]. The sidebar on page 21 provides a brief overview of the metrics mentioned in this article.

Software Product	Implementation Language	Approximate Size	Data Analyzed	Metrics Compared	Best Three Predictors
Registration system	COBOL	80,000 LOC	Run-time failures	9	1. CDM 2. Ordinal scale coupling 3. Fan-in times fan-out
Text-processing utility	C	3,000 LOC	Corrective maintenance	17	1. Referential dependency 2. CDM 3. Fan-out
Patient collaborative care system	C++	82,000 LOC	Corrective maintenance	11	1. CDM 2. Number of clients 3. Fan-in
Electronic file transfer facility	Java	6,000 LOC	Corrective maintenance	14	1. CDM 2. Fan-in 3. RFC

Table 1. *Results of the case studies showing which metrics were the best predictors of run-time failures and corrective maintenance.*

The OASIS Course Registration System

OASIS [3] is a university course registration system developed by Vanderbilt Administrative Systems. It comprises 290 COBOL modules that total approximately 80,000 lines of code. Running on a VAX series computer, the software allows multiple interactive student registration sessions to execute concurrently. Communication between the sessions is accomplished through 18 shared resources; namely, disk files and shared memory segments.

During registration periods, a run-time log is kept that records all OASIS events, including registration transactions and system-trapped failures. The data available for this case study consisted of the source code for OASIS and the run-time log for the software product, as recorded during a single registration period. For each failure, we noted which of the 18 shared resources was involved. We then attributed the failure to any module with access to that resource. Then, as previously explained, for each metric we computed the Spearman rank correlation between the value of that metric applied to each module and the number of failures attributed to that module. The results are shown in Table 1.

The three metrics most highly correlated with the number of run-time failures (the three best predictors of run-time failures) were CDM, ordinal scale coupling, and fan-in times fan-out. These are all coupling-based metrics; intramodule metrics like lines of code and cyclomatic complexity fared poorly as predictors of run-time failures.

The ffortid Text Formatting Utility

Our second case study is ffortid [3], a text formatting utility for UNIX used to format Arabic, Hebrew, and Persian text. It converts ditroff output so that designated right-to-left fonts are properly reversed, letters are properly stretched, and slantable fonts are printed on a slanted base line. The ffortid utility is written in C. It is composed of nine modules that consist of 34 functions that total approximately 3,000 lines of code.

Our study examined data from a maintenance period in which eight distinct enhancements were made to ffortid. The required changes varied in magnitude from a simple command line option change to a more complex extension that would allow slanted fonts. During the maintenance period, the developer recorded daily effort statistics including time spent working on the maintenance project, descriptions of the faults encountered as a consequence of enhancement activities, time spent correcting these faults, and the functions to which the time and faults should be attributed. From this information we were able to compute six corrective maintenance measures. The results were similar for all six measures; for brevity, we report on just one here; namely, the time spent performing corrective maintenance.

Table 1 shows that the best three predictors were referential dependency (a component of CDM), CDM, and fan-out. Again, coupling-based metrics outperformed intramodule metrics.

The Collaborative Care System

Our third case study is a comprehensive patient care management system developed by the Vanderbilt University Medical Center [3]. Designed using object-oriented methods, it is implemented in C++. The product defines 113 distinct classes and consists of 312 modules that total approximately 82,000 lines of code.

Our study examined data from a maintenance period beginning with the original installation of the product and ending with the product's second release. During this period, little or no enhancement was made to the product. Instead, maintenance consisted mostly of modifications made to correct residual faults. The configuration control tool used in the development and maintenance of the product provides accurate statistics regarding the modifications made to the source files over time. For our study, we were therefore able to compute the total number of changes (additions, deletions, or modifications of lines of code) made to each class as well as the total number of times a class was revised.

The results appear in Table 1. Again, CDM outperformed the other predictors of software quality, followed by number of clients and fan-in. In this case study, coupling-based metrics outperformed both intramodule metrics and inheritance-based (object-oriented) metrics.

The Electronic File Transfer Facility

Our fourth case study was with a program called submit, an electronic file transfer facility used in Vanderbilt University engineering courses to allow students to submit class assignments

directly to their instructors for evaluation [3]. Because the hardware platforms used by students vary, Java was chosen as the implementation language for the product. Implemented in a client-server architecture with clients for both UNIX and Windows NT environments, the product defines 29 distinct classes and consists of six packages that total approximately 6,000 lines of code.

In this study, we examined data from a maintenance period beginning with the original installation of the product and ending with the installation of the product's second version. During this period, no enhancements were made to the product. Instead, maintenance consisted mostly of modifications to correct residual faults as well as design flaws with regard to the client-server architecture. The source code was meticulously annotated during maintenance to provide accurate statistics regarding the modifications made to the source files.

The results of computing the Spearman rank correlation between the maintenance data and associated metric values are shown in Table 1. CDM again outperformed the other metrics, followed by fan-in and response for a class. Again, the coupling-based metrics were better predictors of maintenance measures than intramodule metrics (like lines of code or cyclomatic complexity) or inheritance-based metrics (like the depth of the inheritance tree).

Coupling, Faults, Failures, and Maintenance

The obvious question is, why are coupling-based metrics in general (and CDM in particular) such excellent predictors of run-time faults and corrective maintenance? We believe that in most software products, a significant impediment to maintenance is the level of interconnection between modules; that is, the *coupling* between modules.

Consider an arbitrary module M. Let *V* denote the value of a coupling-based metric applied to M. Assume that this coupling-based metric incorporates all possible types of coupling between M and the rest of the product. Then, if a change is made outside M, *V* is a measure of the probability that the change outside M will require a corresponding change within M. In some cases, the need for this change within M will be revealed by the compiler or linker. However, other types of changes may be overlooked, especially when a medium- or large-scale product is developed by a team. Unless the required change is made, a run-time failure may eventually result. This is why coupling-based metrics in general are good predictors of run-time failures.

Turning now to maintenance: to fix a run-time failure requires corrective maintenance. Thus, a metric that can predict where a run-time failure is likely to occur will also be a good predictor of module-level corrective maintenance measures like the number of faults or time to repair faults.

On average, corrective maintenance occupies less than 20 percent of the total maintenance effort [1]. However, coupling also is a good predictor of all other forms of maintenance, including perfective and adaptive maintenance. The reason is that during maintenance the code is changed, and coupling is a measure of the likelihood that a change outside M will necessitate a change within M, irrespective of the reason for that change. That is, the value of a coupling-based metric is a measure of the probability that M must be changed as a consequence of any change to the rest of the product.

Conclusion

We have shown in a set of four case studies that coupling-based metrics like the coupling dependency metric (CDM) are powerful tools for measuring the impact of change. That is, coupling-based metrics are a good way to predict run-time failures and maintenance measures. ◆

About the Authors

Aaron B. Binkley is a senior software developer at Volpe, Brown, Whelan & Co., a San Francisco-based investment banking firm that serves companies in the areas of health care and technology. The research described in this article was performed while he was a graduate student at Vanderbilt University, where he obtained a master's degree in computer science. His primary research interests include software quality metrics and database performance tuning.

Volpe, Brown, Whelan & Co.
One Maritime Plaza
San Francisco, CA 94111
Voice: 415-274-7980
Fax: 415-434-4632
E-mail: aaron_binkley@vbwco.com

Stephen R. Schach is an associate professor of computer science at Vanderbilt University. He also is a software engineering consultant with over 25 years of computer experience, working with industry and giving seminars worldwide on the object-oriented paradigm and software metrics. He has published over 90 refereed technical papers. The fourth edition of his book, *Classical and Object-Oriented Software Engineering*, was published by McGraw-Hill in August 1998.

Vanderbilt University
Computer Science Department
Box 1679, Station B
Nashville, TN 37235
Voice: 615-322-2924
Fax: 615-343-5459
E-mail: srs@vuse.vanderbilt.edu
Internet: http://www.vuse.vanderbilt.edu/~srs/

References

1. Schach, Stephen R., *Classical and Object-Oriented Software Engineering*, 4th ed., McGraw-Hill, New York, 1999 (published August 1998 with a 1999 copyright).
2. Binkley, Aaron B. and Stephen R. Schach, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures," *Proceedings of the 22nd International Conference on Software Engineering*, Kyoto, Japan, April 1998, pp. 452-455.
3. Details of all four case studies can be found in Technical Reports 97–03 through 97–06, Computer Science Department, Vanderbilt University, Nashville, Tenn., 1997 (see http://www.vuse.vanderbilt.edu/~srs/cdm).

Measurement 101

Elizabeth C. L. Starrett
Software Technology Support Center (STSC)

All too often, I read an article that uses terms or concepts I do not understand. The author, usually an expert on the subject, probably assumes the meaning of these terms is obvious and therefore does not explain them. If the terms and ideas appear frequently, I do not understand the article. In this article, I explain some common measurement information to clarify terms used in the measurement articles in this issue of CROSSTALK.

This article is not for experts in measurement. It is simply a list of definitions, examples, and ideas that may be useful to the nonexpert. I will not dwell on how to set up a measurement program or suggest a measurement process because previous *CROSSTALK* articles have done well enough (some of them are listed at the end of this article).

Definitions

When I first became involved with software measurement issues, I thought "metrics" and "measures" were synonymous, but because organizations and individuals define these terms different ways, they may or may not be synonymous. The following definitions used by the STSC are commonly used.

Measure: A standard or unit of measurement—the extent dimensions, capacity, etc.—of any thing, especially as determined by a standard; an act or process of measuring; a result of measurement [1]. Examples of measures include number of defects and source lines of code (SLOC).

Metric: A calculated or composite indicator based on two or more measures; a quantified measure of the degree to which a system, component, or process possesses a given attribute [1]. An example of a metric is defects per thousand SLOC (KSLOC).

Note that two or more measures make up a metric. Realize also that combining two or more metrics gives the information meaning. For example, while measuring 10 defects per KSLOC for a current project, how does one know if this is good, bad, or average? Comparing this metric with a previous project provides a baseline for the data and gives meaning to the metric.

Cost/Schedule Control Systems Criteria (C/SCSC): A Department of Defense (DoD) method established in 1967 to standardize contractor requirements for reporting costs and schedule performance on major contracts and to provide visibility of accomplishments on each contract. Other U.S. agencies have also adopted similar criteria [2]. Many DoD organizations refer to this method simply as "earned value." Establishing a C/SCSC program tends to be complex and expensive, so it is important to know that earned value and C/SCSC do not have to be synonymous—an earned-value process can be implemented in an organization without implementing a formal C/SCSC-compliant function.

Earned Value: A measure of the value of work performed. Earned value uses original estimates and progress to date to show whether the actual costs incurred are within budget [3].

Indicator: A measure or combination of measures that provides insight into a software issue or concept [4]. For example, if an organization considers customer satisfaction to be an issue, defects per KSLOC found by the customer might be a good indicator of customer satisfaction.

Normalize: To cause to conform to a standard. Normalizing data is a process of dividing the numbers back into themselves to leave a percentage instead of actual numbers. For example, an organization that begins tracking its defects per KSLOC comes up with six defects per KSLOC. Every time the organization counts its defects per KSLOC in the future, that number is divided by 6. This gives a relative number that the organization can use to track whether it is improving. It also gives the organization a number that releases minimal sensitive information if an outside organization sees it.

Standard: An accepted measure of comparison for quantitative or qualitative value. To continue the previous example, the standard number of defects per KSLOC delivered to the customer might be set at four defects per KSLOC. Future releases are compared to the standard of four, which provides a relative perspective to the developer and the customer. The use of an industry standard helps the organization understand how it fares in the marketplace. It is important to know that although formally designated standards are not available, informal industry standards have been established over the past several years based on the experience of recognized experts [5].

Threshold: A preset limit at which point action should be taken as a result of the data. Thresholds are established above or below the standard or both. They are often referred to as *upper* and *lower thresholds*. For example, the upper threshold of defects delivered to the customer may be six defects per KSLOC. If the contracting organization receives problem reports from the customer that show the defects in the delivered software are more than six per KSLOC, an investigation will begin to determine why there are so many errors, and the cause will be corrected.

The definition you choose for any of these terms is not as important as agreeing on common definitions for the entire organization. Ensure that everyone understands and uses those definitions consistently.

Typical Measures

When starting a measurement program, the best approach is to reference the organization's strategic plan and use measurements that will indicate if the organization's goals are being achieved. Unfortunately, many organizations do not have a strategic plan. In this case, the first measurements should address the issues important to project managers and customers (most people involved have issues). The following five measures will often be part of the resulting measurement project.

Size

The amount of software a project develops. The two most common ways to measure the size of software are SLOC and function points. One frequently asked question is, "How do I define SLOC?" There is no single answer for this; the most important thing is to clearly and consistently define it for the group being measured. Three common SLOC example definitions are

- The count the compiler gives when it compiles a program.
- Noncommented lines of code.
- "Any code that requires design, code, documentation, and test. This does not count debugging code that will not be delivered to the customer in the final product." [6]

Function points are sometimes used instead of SLOC and sometimes used in addition to SLOC. Function points measure software size by quantifying its functionality [7].

SLOC are preferred by many organizations because they can be easily counted, and everyone can understand a line of code. Function points are preferred by other organizations because the number will remain consistent across languages and platforms and because an accurate count can be determined at the requirements phase.

Effort

Effort is the amount of work required to perform a task. Some example metrics for effort include

- Man-hours per phase of software development (requirements definition, design, code, and test are common lifecycle phases).
- Man-hours per defined set of requirements.
- Man-hours per project.

Schedule

Schedule is the timing and sequence of tasks within a project [3]. The schedule may include tasks, milestones, activities, and phases required for the project.

It is important to note that the duration of a task is not necessarily the same as the effort involved. Two different projects may take the same amount of calendar days to complete but have differing amounts of effort. A project that takes one person working part time five days to complete has an effort of 20 man-hours, whereas another five-day project that employs two people full time will show 80 man-hours of effort.

Cost

Some people consider cost as a combination of the effort and schedule, e.g., 20 man-hours per week for three weeks equals 60 man-hours. This method of calculation is flawed because it does not take into account the differing costs each organization has. A software customer may have two contractors with similar project requirements. The effort and schedule for the two contractors may be the same, but the cost will be different.

Quality

People differ on what defines a quality product. Some may be happy with the quality if there are few defects. Others may care more about how user-friendly the software is; still others will be concerned about how easy the software is to maintain. Following are some examples of quality metrics.

- Defects: Defects per KSLOC, number of defects found per lifecycle phase, number of defects inserted per phase, cost to fix defects, impact of defects on delivered system, cause of defect insertion.
- User friendliness: Response time of system, capability of a system to recover from user errors.
- Maintainability: The ease or difficulty of keeping a system up to date and running. Different organizations have developed processes for scoring the maintainability of software products. The process developed by the Air Force Operational Test and Evaluation Center (AFOTEC) serves as one good example. This process includes a list of questions related to the software. A board is established that reviews the software and decides how well it meets the criteria of the questions. The board then scores the software on a scale of 1 to 6: "1" means that the software is nearly impossible to maintain and "6" that it is easy to maintain.
- Rework: Any effort in reaccomplishing work already deemed complete. Rework effort begins once a defect is found and continues until all the work required to obtain acceptance of the rework is complete [4]. "Already deemed complete" is the area of difference among organizations. Some consider this to be work deemed complete by the programmer, so any code change would be considered rework. Another organization does not consider changes to be rework unless they are changes made after software is released to the customer. The definitions follow the entire range within this time frame.

Crime and Punishment

By law you must collect and act on measurements if you are a government organization. Rather than list every law here, following this article is a copy of Appendix A from the *Air Force Information Technology Investment Performance Measurement Guide*. The appendix is a list of the current laws that require measurements.

Methods

There are numerous effective methods to implement a measurement program, but to list them all is beyond the scope of this article. However, I will mention the Practical Software Measurement (PSM) method. PSM is important because it is sponsored by the Joint Logistics Commanders (JLC) Joint Group on System

Engineering and is gaining acceptance among government organizations. (The JLC comprises members from each of the services who work on issues applicable to all parts of the DoD). PSM was developed as an aid to establish a measurement program. It currently includes a guidebook, training, and a software tool that implements the PSM process. *PSM: A Foundation for Objective Project Management* (typically called the PSM Guide) describes organizations at the beginning of the measurement process, indicating who should be involved and their responsibilities. The guide also suggests different issues that may affect a software project, what measurements would help track those issues, how to collect and analyze the measurements, and suggests how to act on the analysis conclusions. The guide is available from http://www.psmsc.com.

Existing Data

Organizations often ask me for example data of other organizations to which they can compare their data. Unfortunately, most organizations consider this data to be extremely sensitive and rarely release it to the public without normalizing it first. A limited amount of sample data can be accessed from the National Software Data and Information Repository (NSDIR). However, this repository has not been actively maintained for over a year, and the data is approximately two years old. The NSDIR can be accessed at http://nsdir.cards.com/nsdir.

Closing

If you read an article that contains terms you do not understand or hear terms used that leave you confused, please feel free to contact us. We also welcome any editorial comments you may wish to send us on measurement or other software-related issues.

I also ask all measurement experts to keep in mind as you write an article, give a presentation, or talk with customers that your audience is likely not expert—terms and ideas that are intuitive to you may not be intuitive to your audience. ◆

About the Author

Elizabeth C. L. Starrett has been a software engineering consultant for the STSC for five years, where she helps clients improve their software processes. Her most recent duties include leading the STSC Measurement Team. She has spoken at the Software Technology Conference, the Data Reduction and Computer Group Conference, and has been published in *CROSSTALK*. Prior to joining the STSC, she worked for the Air Force with its supporting contractors to develop, document, and test data analysis and test support software for radar and the Peacekeeper missile. She has a bachelor's degree in electrical engineering from Utah State University.

Elizabeth C. L. Starrett
OO-ALC/TISEC
7278 Fourth Street
Hill AFB, Utah 84056-5205
Voice: 801-775-5555 ext. 3059
DSN 777-9730
Fax: 801-777-8069 DSN 777-8069
Internet: starretb@software.hill.af.mil

References

1. STSC Measurement Team, *Measurement Foundation Workshop*, 1994.
2. Fleming, Quentin W., *Cost/Schedule Control Systems Criteria the Management Guide to C/SCSC*, Probus Publishing Company, Chicago, Ill., 1992.
3. *User's Guide for Microsoft Project*, Microsoft Corporation, 1995.
4. *Practical Software Measurement: A Foundation for Objective Project Management*, Ver. 3.1, April 1998.
5. Jones, Capers, *Patterns of Software Systems Failure and Success*, International Thomson Computer Press, Boston, Mass., 1996.
6. Jensen, Randall W., "Estimating the Cost of Software Reuse," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, May 1997.
7. *Function Point Counting Practices Manual*, Dun & Bradstreet Software, Atlanta, Ga., 1994.

Recommended Reading

Methods

1. Grady, Robert B. and Deborah L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.
2. STSC Measurement Team, *Software Metrics Capability Evaluation Guide*, Software Technology Support Center, Hill Air Force Base, Utah, October 1995.
3. Giles, Alan, "Measurement – The Road Less Traveled," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, April 1996.
4. Pitts, David, "Metrics: Problem Solved?" *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, December 1997.

Sample Measures and Metrics

5. Webb, David R. and David Haakenson, "Making Metrics Work Miracles," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, August 1995.
6. Stark, George, "Maintenance Measures," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, July 1997.

Function Points

7. *Function Point Counting Practices Manual*, Dun & Bradstreet Software, Atlanta, Ga., 1994.
8. Heller, Roger, "An Introduction to Function Point Analysis," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, November 1995.
9. Garmus, David, "Function Point Counting in a Real-Time Environment," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, January 1996.

PSM

10. *Practical Software Measurement: A Foundation for Objective Project Management*, Ver. 3.1, April 1998.

This reprint of Appendix A from the Air Force Information Technology Investment Performance Measurement Guide, August 1997 *provides official directives with regard to metrics for Air Force and government organizations.*

Appendix A – Guidance Documents

Chief Financial Officers Act (CFOA) of 1990

The CFOA requires agencies to include performance measurement data in their annual financial statements.
(http://www.npr.gov/library/misc/cfo.html)

Government Performance and Results Act (GPRA) of 1993

The GPRA requires strategic planning and performance measurement in the executive branches of the government. Purposes are to improve federal management, congressional decision-making, service delivery, program effectiveness, public accountability, and public confidence in government. The GPRA requires agencies to develop strategic plans by September 30, 1997, for implementation in fiscal year 1999. The OMB (Office of Management and Budget) has mandated that the plans cover six years and be updated at least every three years. Stakeholders and customers will provide input into the strategic plans. Beginning in fiscal year 1999, agencies will develop yearly performance plans and set performance goals based on their strategic plans. Starting in March 2000, agencies will write annual performance reports, comparing actual performance to goals established in annual performance plans.
(http://www.hhs.gov/progorg/fin/gpraindx.html)

OMB Circular A-11, Part 2: Preparation and Submission of Strategic Plans

This circular provides executive guidance for preparing and submitting agency strategic and performance plans as required by GPRA.
(http://www.whitehouse.gov/WH/EOP/OMB/html/circulars/a011/toc97.html)

Federal Acquisition Streamlining Act (FASA) of 1994

The FASA contains specific requirements for federal agencies to "define the cost, performance, and schedule goals for major acquisition programs" and to monitor and report annually on the degree to which these goals are being met. Agencies must assess whether acquisition programs are achieving 90 percent of cost, performance, and schedule goals and, if not, determine whether to continue the program.
(http://thomas.loc.gov/cgi-bin/query/z?c103:S.1587.ENR:)

Paperwork Reduction Act of 1995 (PRA 95)

The PRA 95 intends to improve the quality and use of federal information; to minimize the cost to the federal government of the creation, collection, maintenance, use, dissemination, and disposition of information; and to ensure that information technology is acquired, used, and managed to improve performance of federal agency missions.

Per PRA 95, agencies must:

- Develop and maintain a strategic information resources management plan that shall describe how information resources management activities help accomplish agency missions
- Develop and maintain an ongoing process to:
  1. ensure that information resources management operations and decisions are integrated with organizational planning, budget, financial management, human resources management, and program decisions;
  2. in cooperation with the agency Chief Financial Officer (or comparable official), develop a full and accurate accounting of information technology expenditures, related expenses, and results; and
  3. establish goals for improving information resources management's contribution to program productivity, efficiency, and effectiveness, methods for measuring progress toward those goals, and clear roles and responsibilities for achieving those goals.

(http://www.os.dhhs.gov/progorg/oirm/pl104-13.txt)

OMB Circular A-130: Management of Federal Information Resources

This circular provides executive guidance on the management of federal IM/IT resources in compliance with PRA 95. Specific requirements include strategic IM/IT planning tying IT investments to agency mission accomplishment and cost/benefit analysis of IT systems throughout the system life-cycle.
(http://www.whitehouse.gov/WH/EOP/OMB/html/circulars/a130/a130.html)

Clinger-Cohen Act (formerly known as Information Technology Management Reform Act [ITMRA]) of 1996
The Clinger-Cohen Act directs that investments in IT support the mission, long-term goals and objectives, and annual performance plan of the department. It mandates that the Secretary of Defense implement performance measurement for all DoD IT programs, projects, and acquisitions.
(http://www.dtic.mil/dodim/cohen.html)

OMB Circular A-11, Part 3: Planning, Budgeting, and Acquisition of Fixed Assets
This circular provides executive guidance on planning, budgeting, and acquisition of fixed assets, specifically IT and NSS-IT, in accordance with GPRA and Clinger-Cohen Act. It requires agencies to identify baseline goals for cost, schedule, and performance for all proposed and ongoing acquisitions, and provides guidance on reporting compliance with these goals to OMB.
(http://www.whitehouse.gov/WH/EOP/OMB/html/circulars/a011.toc97.html)

Executive Order 13011, Federal Information Technology
This order implements the provisions of Clinger-Cohen Act in the executive branch. Besides the specific provisions of Clinger-Cohen Act, the order establishes the Federal CIO Council; creates the Government Information Technology Services Board and the Information Technology Resources Board; and provides additional guidance on the roles of agency CIOs and the use of performance measurement in evaluating IT investments.
(http://www.npr.gov/library/direct/orders/27aa.html)

Executive Office of the President, Evaluating Information Technology Investments – A Practical Guide (OMB Information Technology Investment Guide), November 1995
Provides an analytical framework for linking IT investment decisions to strategic objectives and business plans in the federal organizations.
(http://www.whitehouse.gov/WH/EOP/OMB/infotech/infotech.html)

GAO, Executive Guide: Effectively Implementing the Government Performance and Results Act, (GAO/GGD-96-118), June 1996
Identifies key steps needed to implement GPRA, along with key steps that agencies need to take toward its implementation.
(http://www.gao.gov/special.pubs/gpra.htm)

DoD, Guide for Managing Information Technology (IT) as an Investment and Measuring Performance, February 1997
The guide summarizes the DoD position on IT performance measurement and presents a framework for managing information technology programs as investments rather than as acquisitions.
(http://www.dtic.mil/c3i/cio)

DoD, Information Technology Management (ITM): Supporting National Defense (ITM Strategic Plan), Version 1.0, March 1997
(http://www.dtic.mil/c3i/cio)

Air Force Information Resources Management VISTAS (Air Force Information Resources Management Strategic Plan)
(http://www.cio.hq.af.mil/docs/vistas.htm)

DoD 5000.2-R, Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information System Acquisition Programs
AFI 10-601, Mission Needs and Operational Requirements Guidance and Procedures.
(http://web7.whs.osd.mil/dodiss/publications/pub2.htm)

This document was prepared for Arthur Money, chief information officer (CIO), U.S. Air Force by Andrulis Corporation. For information concerning the project, contact

James Brown
CIO Support Directorate
AFCIC/ITIM
1250 Air Force Pentagon, Room 4A1088E
Washington, DC 20330-1260
Voice: 703-697-3492
Fax: 703-614-4471, -6346
E-mail: brownjd@af.pentagon.mil

Partnership Process for Electronic Warfare Acquisition

Maj. Art Huber and Lt. Col. Jay G. Santee
Office of the Assistant Secretary of the Air Force for Acquisition

The complete version of this article can be found on the CROSSTALK Web site at http://www.stsc.hill.af.mil/CrossTalk/crostalk.html.

The Partnership Process is an acquisition reform initiative that has emerged from the electronic warfare (EW) community. The new process draws on lessons learned from world-class companies to re-engineer EW acquisition. These companies are customer-driven, so the lesson for Air Force acquisition is to respond to the voice of the war fighter by using military worth as the procurement criterion. Top companies maintain open dialog with their suppliers so the Partnership Process emphasizes new ways to foster communication with industry. The best organizations achieve their results through continuous optimization, so we must adopt methods to converge on best solutions.

The new acquisition process can be summarized by six activities (see Table 1) that consistently put superior solutions into the hands of our war fighters as quickly and inexpensively as possible.

These results were achieved through a series of intensive integrated process team meetings that included broad representation from Air Force organizations and the contractor community. Our new process is described in a comprehensive report now available on our Web home page (http://ewio.wpafb.af.mil) and on a CD-ROM available free while supplies last. Currently, our focus is on laying the groundwork for process implementation through a war fighter-led pilot program, the development of an Internet-based training course, and the release of a quick-turn, PC-based decision aid known as the Measures of Effectiveness Tool. The pilot program will demonstrate the application of partnership principles in an area of interest designated by a major command requirements office. The training course will be developed with both government and industry audiences in mind and include the latest reform initiatives (such as the Lightning Bolts) within the holistic view of the partnership. ◆

Table 1. *Measuring military worth forms the foundation for our reformed acquisition process.*

Activity	Innovative Theme
Quantify mission deficiencies.	Base deficiency analysis on war fighter stategy-to-task.
Establish requirements.	Frame the requirement in terms of airspace bought back.
Convey requirements.	Structure Request for Proposal to ask for military worth, not specifications.
Select the source.	Incentivize the contractor to reach beyond thresholds.
Develop the solution.	Continuously optimize the trades to converge to a solution.
Evaluate the result.	Link test and evaluation directly to war fighter needs.

Maj. Art Huber
Voice: 703-588-6515 DSN 425-588-6515
Fax: 703-588-1225 DSN 425-588-1225
E-mail: HuberA@af.pentagon.mil

Y2K from page 10

including the Institute of Electrical and Electronics Engineers conference in St. Louis, Mo. and the DoD Database Colloquium in San Diego. He has also worked other software efforts within the Air Force such as software process improvement. He has a master's degree in business administration from Southern Illinois University and a bachelor's degree in computer science from the University of St. Thomas.

AFCA/ITYO
203 W. Losey Street, Room 1065
Scott Air Force Base, IL 62225-5224
Voice: 618-256-5697
E-mail: thomas.ashton@scott.af.mil

References

1. Voas, Jeffrey, "Certifying Year 2000 'Fixes,'" *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, January 1998.
2. De Jager, Peter, "Throwing Down the Year 2000 Gauntlet," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, January 1998.

READY TO DELIVER from page 15

Possible Uses	Explanation
Maintenance Effort Estimation	Use change rate and rework effort information to estimate required maintenance resources.
Process Improvement	Track common causes of changes to identify process changes that might eliminate rework.
Project Management	Focus management interest and development effort on configuration items, subsystems, or feature areas with high numbers of total or remaining changes.
Delivery, Ship, or Release Decision	Set readiness criteria, e.g., no remaining high-severity problems, maximum number of remaining problems, or maximum defect density, prior to product delivery.
Rework Management	Prioritize software changes according to severity and customer priorities.
Schedule Prediction	Use recent closure and identification rate trends to estimate when the software product will meet test or release criteria.
Test Readiness	Set readiness criteria, e.g., no remaining high-severity problems, maximum number of remaining problems, or maximum defect density, prior to field testing.

Table 1. *Software product maturity uses.*

AFIT/CIGW
c/o AFROTC Det. 025
Arizona State University
Tempe, AZ 85287
Voice: 602-704-0135
Fax: 602-704-0135
E-mail: brian.hermann@asu.edu
Internet: http://www.public.asu.edu/~bgh14/maturity

Jim Russell, a former Air Force captain, served as chief of Software Evaluation Methods, Education, and Automation, Software Analysis Division, AFOTEC at Kirtland Air Force Base, N.M. He was responsible for evaluating Air Force systems for software maintainability, supportability, and maturity. He also led the team that improved current evaluations, researched new evaluation methods, coordinated software training requirements, and headed the office's automated software evaluation efforts.

Russell is a graduate of the Air Force Software Professional Development Program at AFIT, and has a bachelor's degree in computer science from Loyola Marymount University. He is currently working toward a master's degree in engineering management from the University of Colorado.

Russell currently works for Honeywell, Inc. in Phoenix, Ariz.

HQ AFOTEC/SAS
8500 Gibson Boulevard SE
Kirtland AFB, NM 87117-5558
Voice: 505-846-5310 DSN 246-5310
Fax: 505-845-5145 DSN 246-5145
E-mail: sas@afotec.af.mil or jim.russell@acm.org
Internet: http://www.afotec.af.mil

References

1. Hermann, Brian G., AFOTEC Pamphlet 99-102, Vol. 6, "Software Maturity Evaluation Guide," March 1, 1996.
2. Goel, Amrit L. and Brian G. Hermann, "Software Maturity Evaluation: When Is Software Ready for Operational Testing or Fielding?" presented at the Software Technology Conference, Salt Lake City, Utah, April 1997.
3. Foody, Michael A., "When is Software Ready For Release?" *UNIX Review*, March 1995.

Coming Events

'98 Software Engineering Institute Symposium

Dates: Sept. 14-17, 1998
Location: Pittsburgh, Pa.
Subject: This symposium provides a forum to articulate currently applicable practices that software practitioners can use to improve *what* they build by improving *how* they build.
Sponsor: Software Engineering Institute
Contact: Voice: 412-268-5800
E-mail: customer-relations@sei.cmu.edu
Internet: http://www.sei.cmu.edu

Call for Participation: Thirteenth International Forum on COCOMO (COnstructive COst MOde) and Software Cost Modeling

Theme: Software Sizing
Dates: Oct. 6-8, 1998
Location: Los Angeles, Calif.
Subject: This year's forum particularly solicits presentations on software sizing, e.g., object points or other graphical measures, function points, UML-based or other object-oriented measures, alternative sizing techniques, comparison of software size measures, and usage and calibration of size measures in the COCOMO II submodels and other cost models.
Sponsor: University of Southern California Center for Software Engineering and the Software Engineering Institute
Contact: Jennifer Browning, Center for Software Engineering, University of Southern California, Los Angeles, CA 90089-0781
Voice and Fax: 213-740-5703
E-mail: browing@sunset.usc.edu

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*** 801-775-5555 ext. 3027 graphics@stsc1.hill.af.mil
Associate Editor	***Lorin J. May*** 801-775-5555 ext. 3026 backtalk@stsc1.hill.af.mil
Editorial Assistant	***Bonnie May*** 801-775-5555 ext. 3022 customer_service@stsc1.hill.af.mil
Features Coordinator	features@stsc1.hill.af.mil
Customer Service	801-777-8045 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: 801-777-8045 DSN 777-8045
Fax: 801-777-8069 DSN 777-8069

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

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

Reprints and Permissions: 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. The STSC maintains a Gopher server at gopher://gopher.stsc.hill.af.mil. Its ftp site may be reached at ftp://ftp.stsc.hill.af.mil. The Lynx browser or gopher server can also be reached using telnet at bbs.stsc.hill.af.mil or by modem at 801-774-6509 or DSN 775-3602. Call 801-777-7989 or DSN 777-7989 for assistance, or E-mail to portr@software.hill.af.mil.

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

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

Really Bad Metrics Advice

According to my data, roughly 122.45 percent of this journal's 347,583,712 readers need some sharpening up on how to effectively collect and use metrics. There is less than a 0.0345 percent chance that this column will help.

Q: I'm a manager who believes in keeping metrics simple, which is why I've limited the number we collect to 62. But I also want to simplify their collection—do you know where I can find timecard readers designed for bathroom stalls?

A: Try voice print-activated stalls with timed door locks. But first, are you really trying to collect *62* metrics? *62?* [snicker snort chortle] You're obviously clueless about the "KISS" principle: *Keep It Stupefyingly Strenuous.* You can collect *a lot* more than 62 different metrics. The accepted rule of thumb for the number of metrics you can reasonably work with is this: "Seven, plus or minus the square of the number of door knobs in your home." Remember, if something *can* be measured, it *must* be measured, and all metrics are equally critical.

Q: I feel vindicated. Now I can introduce additional metrics for every obscure area of our process improvement model. Naturally, I plan to drop the whole wad as an enforced edict and then make myself unavailable for a few weeks.

A: Bravo! But be sure you don't overcomplicate things by defining every minute detail, such as data integrity standards or what you plan to do with the data. People learn nothing from constant handholding. Your job is to sit back and wait for those reliable numbers to start pouring in.

Q: Great! What do you suggest I do with all that data?

A: What should you *do* with the data? Do? That question implies that metrics are a means to some end. Don't waste resources—time spent analyzing metrics is time that could have been spent collecting even more metrics.

Q: My boss keeps asking for data on stuff I don't think can be quantified—and it's often common sense stuff he could just ask us! Aren't metrics just a big sham?

A: Shhh! You're right, metrics are actually an extensive conspiracy—but an extremely helpful one. When people want to make decisions based on "facts" rather than "opinions," you need metrics to push your personal agenda under the guise of unassailable objectivity. Perception is everything:

Politicized emotional drivel: "Let's try my approach. Her plan isn't working."

Objective insight: "A consumptive analysis of my plan projects a 84.67 percent increased density of pro-active rationals within six months. However, her key preambulatory vindicators are creating a 24.38 percent downward sloping polymorphic trend. Plus, she wears really cheesy business suits."

Q: But what if I don't know how to collect and project those kind of numbers?

A: Then you're in the same boat as the people who want to see your metrics. This is the whole key to effectively utilizing metrics: They don't exist to *uncover* reality—they're for *creating* whatever reality suits you.

Q: I lead a project with a beautifully simple metrics program that consists of two critical measurements: How many days past the deadline we are, and how many dollars over budget. But lately I've had the nagging feeling that I'm not getting enough mileage from these metrics. Is there any way I can use metrics to promote dishonesty, infighting, and poor work habits?

A: Certainly. Once you've worn down employees with coma-inducing quantities of metrics that have no perceivable link to any business objectives, pick one favorite, such as lines of code, then base penalties and rewards on it. Resulting competition will discourage teamwork and will lead to ineffective work and "creative" reporting practices among some employees. Their skewed metrics will give you great overall numbers, which you can then use to dazzle your superiors.

Q: "Great" numbers draw scrutiny. How about "barely exceeding expectations"?

A: Fortunately, herpetological analysis indicates a 98.65 percent propensity toward established parameters, regardless of iambic deviance from ergonomics.

Q: Huh?

A: If you tell your employees what final numbers you want to see, no matter how absurd, they'll manage to deliver them without even breaking a sweat.

Q: And you don't think anyone will audit my metrics for accuracy?

A: You can bet 97.387 percent of the farm on it. – Lorin May

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