Test Drive Your Software

Tracy Stauder
Managing Editor

Whether you are in the market to purchase a new car, home, computer, television, or other high-priced item, the item's quality is sure to affect your buying decision. Everyone wants high quality, but can we afford it? In today's society, we have been geared to expect higher quality when paying higher prices. Top-model products often demand top prices. Now and then, we do get lucky and get a *Cadillac* at a *Geo* price, but those instances are few.

Since quality and price seem to correlate, many of us do our homework before spending large sums of money on what we hope are quality products. We are careful to measure quality through our own individual methods. One method is usually related to our senses. For instance, we want to see how a new coat fits and to hear the sound of a new stereo before we determine if the quality is sufficient to meet our needs. Our other quality-measuring methods include research and testing. Before we buy a new computer, we research the technology details of memory and speed. And before we purchase a new car or truck, we want to test drive it to see how it handles. In any case, for big-ticket items in our everyday lives, we generally pay much more attention to the details and to the quality before we feel comfortable in making our buying decision.

I see similarities in the software acquisition environment. Software acquirers want high-quality software if they are paying millions of dollars for it. And because of this, software acquirers are highly interested in measuring the quality of software products. They will monitor the software developer to ensure software test plans and test engineering processes are in use. And when possible, they will "test drive" the software to see if their requirements are satisfied. With mission-critical software, the requirement must be "crash free."

Software acquirers should also look for developers who embrace defect prevention and defect tracking processes. As Bala Subramaniam reports in his article (page 3), effective defect tracking can enhance software quality while reducing project costs.

Consider the Software Engineering Institute Capability Maturity Model definition for *quality*.

> "The degree to which a system, component, or process meets specified requirements; or the degree to which a system or process meets customer or user needs or expectations."

Whether buying or developing software, many are faced with measuring this degree of quality.

Software developers should be eager to show the customer that they are quality conscious. Software developers need to assure the customer that defect prevention methods, such as peer reviews and test engineering, are employed in their development processes. Test engineering can help verify that requirements are satisfied at each development phase.

Software development teams that work together with their customer to set quality goals will best satisfy the end-user needs and desires. So, whether you are on the buying or the producing side of the software equation, software quality assurance through defect prevention and testing is a must. Perhaps you can take your software for a test drive today. ◆

Call for Articles

If your experience or research has produced information that could be useful to others, *CROSSTALK* will get the word out. We welcome articles on all software-related topics, but are especially interested in several high-interest areas. In a future issue, we will place a special, yet nonexclusive, focus on

Software Best Practices
October 1999
Article Submission Deadline: June 1, 1999

Managing Technological Change
November 1999
Article Submission Deadline: July 1, 1999

We will accept article submissions on all software-related topics at any time; issues will not focus exclusively on the featured theme.

Please follow the *Guidelines for* CROSSTALK *Authors*, available on the Internet at http://www.stsc.hill.af.mil. Send articles to

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

Or E-mail to features@stsc1.hill.af.mil. For more information, call 801-775-5555 DSN 775-5555.

Effective Software Defect Tracking

Reducing Project Costs and Enhancing Quality

Bala Subramaniam
ISSRe Systems, Inc.

The costs of defective software can be as high as 50 percent of the investment in software development. Yet, the potential to improve software quality and reduce project cost is enormous. Software defect tracking can be an effective means to achieve quality at less cost. However, defect tracking is commonly misunderstood, incorrectly implemented, and often seen as an impediment and cost to the organization. This article discusses the quality costs of defective software and provides a working model to implement an effective software defect tracking system within an organization.

Defect tracking is sometimes written off as boring, repetitive, and unglamorous. Even effective defect tracking is often viewed as an unnecessary cost that impedes schedules. Yet, defect tracking is one of the most critical components of the software development and the quality assurance efforts.

When implemented well, defect tracking greatly reduces overall project costs and improves schedule performance. As a critical component in improving software quality, the potential paybacks for such processes are enormous. A Hewlett-Packard quality program reduced software errors by 75 percent and cut development time 20 percent. An Air Force systems group reports that every dollar invested to improve quality has a conservative return of $7.50.

To effectively track and manage software defects also improves customer satisfaction, creates higher productivity and quicker delivery, and leads to better operational reliability and improved morale. On the other hand, a mismanaged software defect tracking program may indeed be an unnecessary cost.

Software defects take different names in different organizations, e.g., errors, issues, bugs, defects, or incidents. Whatever they are called and whatever form they take, defects can have an astounding impact on the development phase and can continue to haunt the product through its maintenance phase.

The costs to fix software defects are high, especially if fixing requires developers to re-familiarize themselves with months-old work or if someone other than the original developer is doing the fixing. Costs also increase exponentially while moving further along the software development lifecycle. Studies at IBM demonstrate that compared to catching defects before or during coding, it is 10 times more costly to correct an error after coding and 100 times more costly to correct a production error.

Software Quality Costs

A 1996 study by The Standish Group reported that U.S. businesses invest about $250 billion in software development annually, yet a great many of these projects fail because of cost overruns. One of the significant components of project costs is software quality cost. One estimate put the cost of a single post-release defect to a large organization as high as $20,000 to $40,000.

Software quality costs are the costs associated with preventing, finding, and correcting defective software. Following are three useful definitions of quality costs [1].

Prevention Costs. These are costs of activities specifically designed to prevent poor-quality software, e.g., costs of efforts to prevent coding errors, design errors, additional document reviews to reduce mistakes in the user manuals, and code reviews to minimize badly documented or unmaintainably complex code. Most of these prevention costs do not fit within a typical testing group's budget. The programming, design, and marketing staffs spend this money.

Appraisal Costs. These are costs of activities to find defects, such as code inspections and software quality testing. Design reviews are part prevention and part appraisal. Formulating ways to strengthen the design is a prevention cost, whereas to analyze proposed designs for potential errors is an appraisal cost.

Failure Costs. These are costs that result directly from poor software quality, such as the cost to fix defects and the cost to deal with customer complaints. Failure costs can be divided into two main areas:

- Internal failure costs: Costs that arise before the product is delivered to the customer. Along with costs to find and fix bugs are costs associated with wasted time, missed milestones, and overtime needed to get back on schedule.
- External failure costs: Once the software is delivered to the customer, poor-quality software can incur customer service costs or the cost to distribute a patch for a released product. External failure costs are huge; it is much cheaper to fix defects before shipping the defective product to customers. If a product has to be shipped late because of bugs, the direct cost of late shipment includes the lost sales, whereas the lost opportunity cost of the late shipment includes the costs of delaying other projects while everyone finishes the one that is error-ridden.

User interface defects are often treated as low priority and are fixed last. This can be a mistake. Product screens may be required for effective marketing and documentation. This can

result in increased costs in nondevelopment areas, lost marketing opportunities, and contractual penalties. Unfortunately, numerical estimates of lost opportunity costs and delays are difficult to make and can be controversial [2].

All the above software quality costs contribute to the total cost of poor-quality software to the organization. In aggregation, the total cost of software quality may be presented as follows:

Total Cost of Quality = Prevention + Appraisal + Internal Failure + External Failure.

What Is a Defect?

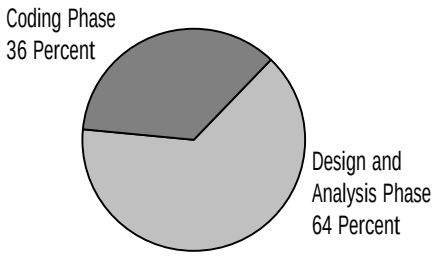
Defects are commonly defined as "failure to conform to specifications," e.g., incorrectly implemented specifications and specified requirement(s) missing from the software. However, this definition is too narrow. Discussions within the software development community consistently recognize that most failures in software products are due to errors in the specifications or requirements—as high as 80 percent of total defect costs [3]. Other studies have shown that the majority of system errors occur in the design phase [4]. Figure 1 represents the results of numerous studies that show approximately two-thirds of all detected errors can be traced to the design phase.

I recommend a broader definition of defect: *variance from a desired attribute.* These attributes include complete and correct requirements and specifications, designs that meet requirements, and programs that observe requirements and business rules.

Implementing an Effective Defect Tracking Process

Software quality assurance departments can play a catalytic role in implementing an effective defect tracking process. A survey conducted in 1994 by the Quality Assurance Institute found that a mere 38 percent of the organizations had formal software defect management processes, whereas 25 percent of the survey participants said their organizations lack consistent testing standards and procedures [5]. The survey also reported that although 60 percent of organizations had testing standards and procedures, some organizations admitted they were out of date and not followed. Recent surveys, nonetheless, suggest that more companies are now striving to improve their software development process through early defect identification, minimizing resolution time, hence reducing project costs.

Figure 1. *Origin of software errors across industry.*



Effective defect tracking begins with a systematic process. A structured tracking process begins with initially logging the defects, investigating the defects, then providing the structure to resolve them. Defect analysis and reporting offer a powerful means to manage defects and defect depletion trends, hence quality costs.

Integrate Software Development and Defect Tracking

Traditional approaches place testing immediately before implementation. Typically, testers receive a low-quality product at the tail end of development when there is tremendous pressure to deliver, even if the software is plagued with defects. For early defect detection and resolution to take place, defect tracking and software development efforts should begin simultaneously. It will solve a multitude of problems downstream.

Defect tracking must be implemented throughout the development lifecycle. On projects I have managed or worked on, this has always lead to fewer release defects; however, such organizational foresight is rare. The Sentry Group reported that 62 percent of all U.S. organizations do not have a formal quality assurance or test group. The report also added that a large majority of these organizations place a much higher priority on meeting schedule deadlines than producing high-quality software [6].

The Different Phases of Defect Tracking

Successful verification throughout the development cycle requires clearly defined system specification and software application business rules.

Requirements phase. Defect tracking focuses on validating that the defined requirements meet the needs and the users' expectation about functionality. Sometimes, system-specific constraints would cause the deletion of certain business requirements.

Design and analysis phase. Efforts should focus on identifying and documenting that the application design meets the business rules or field requirements as defined by the business or user requirements. For example, does the design correctly represent the expected user interface? Will it enforce the defined business rules? Would a simpler design reduce coding time and documentation of user manuals and training? Does the design have other effects on the reliability of the program?

I experienced the downstream cost of a specification error when working on one of two groups of developers who were programming complementary parts of a data-bridging program. Coding was well under way when incomplete system specifications caused transfer of data on the bridge to fail. The failure was not due to coding errors but to specification errors that were translated into program codes. Had the deficiency been discovered before coding began, we could have saved the substantial time and money required to repair the programs.

Programming phases. Defect tracking must emphasize ensuring that the programs accomplish the defined application functionality given by the requirements and design. For example, has any particular coding caused defects in other parts of the application or in the database? Is a particular feature visibly wrong?

Maintenance and enhancement phases. During the maintenance phase,

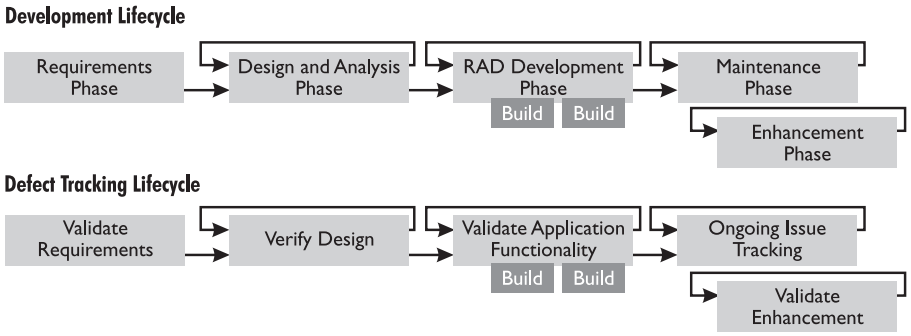


Figure 2. *Defect tracking running parallel to development lifecycle.*

effort is spent tracking ongoing user issues with the software. During enhancement phases (there could be multiple releases), defect tracking is focused on establishing that the previous release is still stable when the enhancements have been added. Figure 2 represents the philosophy of defect tracking throughout the software development process.

Introduce Defect Tracking Early

It is not difficult to introduce tracking early into the development process—it fits well into current software development processes. Today's rapid application development (RAD), the dominant approach in client-server software projects, focuses on shortened development schedules. This software development method provides early review points, delivered as "builds" or iterations of development, to ensure that requirements are met. Such a process clearly lends itself to early defect tracking, which can shadow development (see Figure 2). Each build can receive verification that it meets the defined requirements; if not, defects can be reported and resolved quickly and relatively inexpensively while the software is still "pliable." The same defect reported later in the development process may require a major "surgery" to the software product; hence, it will be more costly. Front-end defect tracking costs much less than waiting until the end.

> "Quality comes not from inspection, but improvement of the development process."
>
> – W. Edward Deming. [7]

An Effective Defect Tracking Process

To merely integrate defect tracking into the development process is not enough. A clearly defined defect tracking process is needed to ensure defects are handled in an organized manner from discovery through resolution. Components of this process are described in the sections that follow. This process is progressive—defect evaluation cannot be successfully performed if the earlier components (such as describing defects and prioritizing defects) were not implemented.

Defect Repository

Once a defect has been discovered, the important first step is to log the defect into a defect-tracking database or repository. When a defect is logged, it must be fully described so that it can be reproduced during debugging, prioritized based on its severity, and have resources assigned for its resolution. Defects have a number of other attributes that should be recorded, such as

- Defect number.
- Date.
- The build and test platform in which it was discovered.
- The application requirement or business rule to which it relates.
- Any supplementary notes.

It also is important that the repository offer a means to track the "life" of the defect (the resolution status) and historically report on all defects discovered and logged for the project. It pays to have this system online and available to all development staff so that the assigned parties can update the resolution progress for the defect status.

Defects Described

Your organization's defect reporting procedures should require that details about each software defect be recorded when the defect is discovered, including a description, symptoms, sequence of steps to re-create it, and severity. Defects are of various types:

- Interface defects include incorrectly working menu items, push buttons, and list boxes.
- Navigational defects could be described as a window not opening when moving from one interface screen to another.
- Functionality defects could be incorrect calculation of salaries in a payroll system.

Do not merely log, "Adding new customer window does not work." A detailed description, such as, "The 'Save' button on 'Add New Customer' window does not work," would give the developer adequate information to go straight to the specific problem and repair it. This saves time and unnecessary interruption for the developer to research the defect thus reducing the overall project cost.

Defects Prioritized

Once a defect is logged and described, appropriate resources must be allocated for its resolution. To do this, the defect must be analyzed and prioritized according to its severity. Each defect is given a priority based on its criticality. Usually, it is practical to have four priority levels:

- Resolve Immediately.
- High Priority.
- Normal Queue.
- Low Priority.

A misstatement of a requirement or a serious design flaw must be resolved immediately, before the developer translates it into codes that are implemented in the software—it is much cheaper to amend a requirement document than to make program code changes. The wrong font size for a label may be classified as "Low Priority."

The critical path for development is another determinant of defect priority. For example, if one piece of the functionality must work before the next piece is added, *any* functional defects of the

first piece will be given the "Resolve Immediately" priority level. On one project I worked on, a query engine retrieved transactions matching user-specified criteria upon which further processing was performed. If the query engine had been defective, no further development (or testing) would have been practical. Therefore, all functional defects of the query engine were prioritized as "Resolved Immediately."

The urgency with which a defect has to be repaired is derived from the severity of the defect, which could be defined as follows:

- Critical.
- Important.
- Average.
- Low.

A defect that prevents the user from moving ahead in the application—a "show stopper"—is classified as "Critical," e.g., performing an event causes a general protection fault in the application. Performance defects may also be classified as "Critical" for certain software that must meet predetermined performance metrics. If the user is able to formulate work-arounds where there are defects, these defects may be classified as "Average." An overly long processing time may be classified as "Important" because although it does not prevent the user from proceeding, it is performance deficiency. Defects with severity "Average" will be repaired when the higher-category defects have been repaired and if time permits. Certain graphical user interface defects, such as placement of push buttons on the window, may be classified as "Low," since this does not impede the application functionality. Although defect priority indicates how quickly the defect must be repaired, its severity is determined by the importance of that aspect of the application in relation to the software requirements.

Structured Resolution

The defect tracking system also must ensure that the defect progresses in an appropriate sequence from discovery through resolution. Each defect is also given appropriate status; for example, a new defect is given the status of "Open," and a defect under repair would have the status of "Assigned."

As repair work progresses, the status of defects is updated to reflect its state in the resolution process. A defect that has been repaired will be submitted to the testing team through formal change control to be verified again. Only if the fix passes the regression test will it be accepted and the defect assigned a status of "Closed." Other defect statuses could include "Deferred," if the defect is not to be fixed for the current release but may be resolved in a subsequent release or "Enhancement," if a feature that is not part of the requirements has been suggested, and may be reviewed as an enhancement for later releases.

Communication

An effective defect tracking system must allow communication of the software's defects, status, or changes to members of the development team and all others concerned. This has become an increasingly crucial element because people working on the same project may not only work in different parts of a building but also may even work in a different state for a different organization. Without an effective means to communicate defects, defect tracking—and consequently achieving software quality—would be a nightmare.

E-mail is an efficient vehicle to expedite informing software engineers and all concerned of defects as they are discovered. Software engineers could then perhaps access an online defect repository as they receive the E-mail on new and existing defects. Similarly, E-mail also serves as a reply medium to inform testers that a defect has been repaired. Some defect tracking repositories, e.g., one set up in Lotus Notes, facilitates built-in communication features that can be used by both software engineers and testers.

Commercially available defect tracking software, e.g., AutoTester and SQA Team Test Software, are more sophisticated in communicating defects and their status to individuals or as a batch. They also automatically inform respective development staff and management of defects as they are discovered.

Although E-mail provides a means to convey information about defects between the development and testing team, regular formal defect tracking meetings also help keep a close eye on the number, types, and nature of defects found, which may indicate how software quality is progressing through the resolution stage.

Defect analysis is discussed in more detail in the "Reporting" component of this defect tracking process. If the testing and development teams must work hand-in-hand toward achieving software quality, there must be continuous communication between them. Informal or verbal communication between these teams is inadequate.

Continuous Defect Resolution

It costs much less to resolve defects as soon as they are discovered—do not merely accumulate a list of defects to fix later. For example, in my current project, the software product is undergoing two transformations: The entire application architecture is being revamped and enhancements are being implemented for the next release. Revamping the architecture changes the fundamental "backbone" of the application in question, which is in itself a complex task. We have three categories of defects:

- The existing list of yet-to-be-resolved defects from the original application.
- Defects that would come about as a result of revamping the architecture.
- New defects contributed by the new enhancements.

We have divided the project into smaller deliverables and implemented defect tracking for each deliverable. If resolution were to be delayed until later, the mere complexity of the various deliverables would present an inordinate amount of challenge to resolve defects. Moreover, different software engineers are working on different deliverables and different tasks within each deliverable. At later stages, it would become a mammoth effort to merely identify and assign defects to the respective software engineers. Additionally, when they have been assigned defects to repair, the engineers have to remember what they implemented in the codes perhaps months

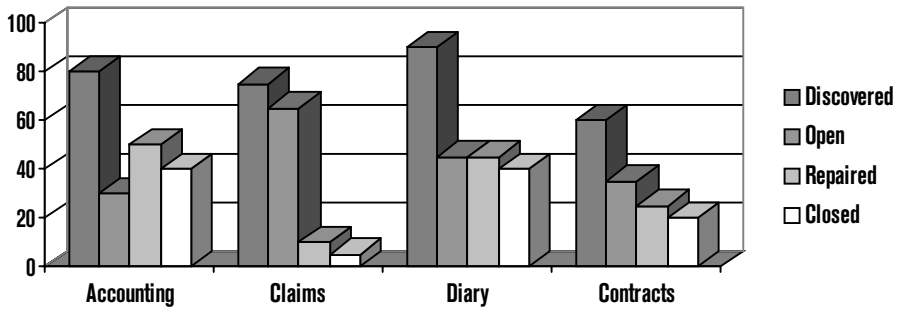


Figure 3. *Resolution efforts for an accounting system.*

earlier. This will incur expensive investigation time.

The best time to resolve defects is when they are discovered. This is especially true in a RAD environment, where the application is developed through several iterations or builds. Each build has an incremental amount of application functionality and related coding. Any defects discovered in a particular build should be referred to developers immediately for resolution. The functionality added in the most recent build and related program codes are still fresh in the developers' minds, which leads to faster investigation of the root cause of the defects, and therefore more efficient resolution efforts. To defer defect resolution until later in the development cycle wastes time and resources.

Defect Evaluation and Analyses

Most organizations consider it essential to constantly monitor and evaluate their performance, and this key practice is especially critical in defect removal. The overall success of your project largely hinges on effective defect resolution, so you need to know your defect removal status and the cost of achieving quality. For example, a defect trend analysis will indicate the number of defects discovered over time. This analysis may even be further subdivided for defects by status, functionality, severity, etc. Defect age analysis suggests how quickly defects are resolved by category.

The type and extent of the defect evaluation and analyses may be determined by the organization's cost objectives and delivery schedules. Following are a few suggested analyses that may be applicable to most software projects. The following measures need to be determined to analyze defects (or those chosen as part of an organization's defect analysis strategy).

- Defect status vs. priority.
- Defect status vs. severity.
- Defect status vs. application module.
- Defect age.

The above information will not be available if the earlier steps of adequately logging defects were not implemented as part of the defect tracking process. By comparing these measures from the current iteration to the results from the analysis of previous iterations, one can get an indication of defects trends, which are discussed further in the following two subsections.

Defect Evaluation. Although the evaluation of test coverage provides the measure of testing completion, an evaluation of defects discovered during testing provides the best indication of software quality. By definition, quality is the indication of how well the software meets a desired attribute. So, in this context, defects are identified as "variance from a desired attribute."

Defect evaluation may be based on methods that range from simple defect counts to rigorous statistical modeling. Rigorous evaluation can include forming a model (or setting goals) about discovery rates of defects, then fitting the actual defect rates during the testing process to the model. The results can be used to estimate the current software reliability and predict how the reliability will grow if testing and defect removal efforts continue. However, because the field's current lack of a scientific model and resources dedicated to perform such evaluations (or a tool to support them), an organization should carefully balance the cost of rigorous evaluation with the value it adds.

Defect Analysis. This means analyzing the distribution of defects over the values of one or more parameters associated with a defect. Defect analysis provides an indication of the reliability of the software. Four main defect parameters are commonly used for defect analysis:

- Status: the current state of the defect (open, being repaired, closed, etc.).
- Priority: the relative importance of addressing and resolving this defect.
- Severity: the relative impact of this defect to the end-user, an organization, third parties, etc.
- Source: what part of the software (such as a module) or requirement this defect affects.

Defect counts can be reported in two ways: (1) as a function of time, resulting in a defect trend diagram or report and (2) as a function of one or more defect parameters (like severity or status) in a defect density report. These types of analysis provide a perspective on the trends or distribution of defects that reveal the software's reliability.

Defect trends follow a fairly predictable pattern in a testing cycle. Early in the cycle, the defect rates rise quickly. Then, they reach a peak about midstream, in an adequately staffed test project, and fall at a slower rate over time. The project schedule can be reviewed in light of this trend. For instance, if the defect rates are still rising in the third week of a four-week test cycle, the project is clearly not on schedule. Other instances where the rate of closing defects is too slow (experience rated) might indicate a problem with the defect resolution process; for example, resources to fix defects or to retest and validate fixes might be inadequate.

This is an important aspect of software project management: to ensure that software quality is progressing within the planned delivery schedule. Figure 3 displays defect status by software module. In each software module, a discovered defect is given a status of Open and assigned resources for fixing.

In Figure 3, resolution efforts for accounting appear to be good, because

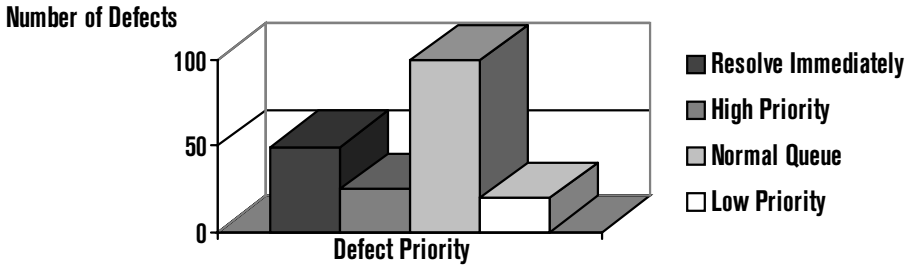


Figure 4. *Defect distribution.*

there are more defects being repaired than left open. Retesting efforts seem to be adequate, because defects closed are not far behind defects repaired. However, although there is a similar retesting effort for the claims module, there are far too many defects open, indicating that additional resources may be required for this module. The trend for the diary module suggests that both defect repair and retesting efforts are progressing well. The defect trend for the contracts module also shows that defect resolution is progressing well.

An organization could set quality criteria for how the distribution of defects over priority levels should look, e.g., "No critical defects should stay open for more than one week." It would be expected that defect discovery rates would eventually diminish as testing and fixing progresses. A threshold can be established below that in which the software can be deployed.

Defect counts can also be reported based on the source, allowing detection of weak modules and "hot spots." Parts of the software that must be fixed repeatedly indicate a fundamental design flaw. In my current project, this type of analysis helped us come to conclude that the application architecture technology needed to be revamped. The originally chosen architecture, although technically superior, was more complicated and made the application extremely delicate to changes or defects "fixes."

Defects included in an analysis of this kind have to be confirmed defects. Not all reported defects turn out to be flaws; some may be enhancement requests, out of the scope of the project, or describe a previously reported defect. These defects must be reclassified as such. However, there is value in analyzing why many duplicates or unconfirmed defects are being reported.

Defect Reporting

Defect evaluation and analysis have to be reported in a useful form to those who make decisions about resources, costs, and delivery schedules. Although each organization may want to produce different reports and different forms, there are three classes of reports:

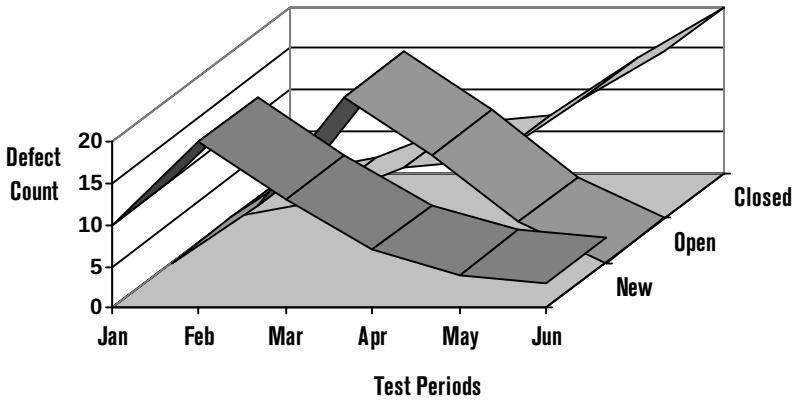
- Defect density or distribution reports allow defect counts to be shown as a function of one or two defect parameters. Using the priority parameter, defect distribution may be represented as shown in Figure 4.
- Defect age reports are a type of defect distribution report that shows how long a defect has been in a particular state, such as Open. In any age category, defects can also be sorted by any other attribute, such as Owner (developer assigned to repair defect).
- Defect trend reports show defect counts by status (New, Open, or Closed) as a function of time. The trend reports can be cumulative or noncumulative and help management identify defect rates by status thus providing an indication of how well the software quality is progressing through the project cycle. Figure 5 represents a typical defect trend report.

In Figure 5, the number of new defects peaked in February. Lagging behind new defects by about a month, the number of open defects was the highest in March. The defect-fixing efforts appear to be consistent throughout the project, closing all defects by June. Before an organization can produce the type of reports discussed here, a defect repository must have been established to support such analysis, i.e., provide for logging of defect description, status, priorities, etc., outlined in the "Defects Described" section of this article. To be successful, all pieces of the defect tracking process must be embraced.

Making This Approach Work

It is unwise to try to achieve too much too fast. Change is the most difficult concept to grasp or implement. The effective defect tracking model discussed here not only may call for a fundamental change in your software development process but also may require a broader concept and definition of defects and the tools to manage them. Depending on the capability of the development and quality assurance process, an organization may not want to attempt the total defect-tracking model all at once. For lower maturity organizations, an incre-

Figure 5. *Defect trend report.*



mental adoption of the recommendations would be more successful.

A good start is to set up a simple defect-tracking repository that implements defect description, status, priority and severity, and communication. Expand that list later to include defect evaluation and analysis and reporting. This ensures that the required defect data is captured as a minimum so the organization can build on this model.

Next, the process should be widened to embrace the broader definition of a defect, and the concept of implementing defect tracking across the development process. The implementation of an effective defect tracking process should be taken through levels of maturity, which is a topic for an entire article. Once the model and process is applied to one project successfully, it can be implemented across the organization.

Conclusion

Effective defect tracking strongly contributes to enhancing software quality and reducing development project costs. Using the broader definition of a defect ensures that not only are resultant errors or nonconformance to requirements discovered but also variance from a desired attribute, including incomplete requirements, takes place. Searches for such defects can then take place across all software development phases.

By "shadowing" the software development process, defect tracking helps you identify and report potential software problems early and acts as a catalyst for problems to be addressed. By facilitating discovery of defects earlier in the development cycle, effective defect tracking is a critical key to lower costs, enhanced software quality, and reducing overall project cost. However, to achieve this requires a fundamental change in the ideology behind quality assurance and the software development process as well as the introduction of the necessary tools to track and manage defects. The defect-tracking model discussed in this article will be useful for organizations moving in this direction. Careful planning and phased adoption of this model can make this approach a powerful software quality strategy. ◆

About the Author

Bala Subramaniam is director of quality assurance at ISSRe Systems, Inc., in New York. He has 15 years managerial and technical experience, during which he has worked for medium and large software companies including IBM. His special interests include the definition and implementation of quality assurance methods and software process improvement programs. He also is experienced in designing effective automated test methods to test complex mission-critical software functionality and business cycles. He has a master's degree in business administration (finance) from Birmingham Business School in Great Britain and is a certified software test engineer (Quality Assurance Institute).

ISSRe Systems, Inc.
200 Business Park Drive
Armonk, NY 10504
Voice: 914-273-7777
Fax: 914-273-7796
E-mail: balas@issre.com

References

1. Campanella, J., ed., *Principles of Quality Costs*, ASQC Quality Press, 1990.
2. Juran, J.M. and Frank M. Gryna, *Juran's Quality Control Handbook*, 4th ed., McGraw-Hill, New York, pp. 4.9-4.12.
3. ANSI/IEEE Standard 982.1-1988, *IEEE Standard Dictionary of Measures to Produce Reliable Software*, Institute of Electrical and Electronics Engineers, p. 13.
4. Perry, W., "Structured Approach to Testing," *Effective Methods for Software Testing*, John Wiley & Sons, New York, 1995.
5. Perry, W., "1994 Survey Results on Software Testing," *Effective Methods for Software Testing*, John Wiley & Sons, New York, 1995.
6. *Automated Software Quality Directions*, The Sentry Group, February 1998.
7. Deming, W. Edward, *Out of the Crisis*, MIT Press, Cambridge, Mass., 1982.

INCOSE '99

The Ninth Annual International Symposium of the International Council on Systems Engineering

"Sharing the Future"

June 6 – 10, 1999
Brighton, England

INCOSE '99 will be *different* – a new continent, a fresh perspective, and coverage of emerging issues.

INCOSE '99 will be *the same* – the world's largest gathering of systems engineering professionals with thought-provoking, relevant papers; up-to-the-minute briefings; and a wide range of tutorials.

To find out more about Brighton, visit http://www.brighton.co.uk. For further details on the conference, refer to the INCOSE '99 Web site.

"Brighton" your systems engineering in '99!

Cass Jones
Conference Manager
7916 Convoy Court
San Diego, CA 92111
Voice: 619-565-9921
Fax: 619-565-9954
E-mail: pcminc@pcmisandiego.com
Internet: http://www.incose.org.uk

Making Requirements Management Work for You

Alan M. Davis, *Omni-Vista, Inc.*
Dean A. Leffingwell, *Rational Software, Inc.*

Requirements are capabilities and objectives to which software must conform and are the common thread for all development activities. Requirements management is the process of eliciting, documenting, organizing, and tracking changing requirements and communicating this information across the project team. Implementing a requirements management effort ensures that iterative and unanticipated changes are maintained throughout the project lifecycle. Without these measures, high-quality software is difficult if not impossible to achieve.

Identifying Requirements – How Will I Know One When I See One?

Requirements begin their lives when first elicited from customers or users. Elicitation may occur using any of a variety of techniques such as interviews, brainstorming, prototyping, questionnaires, and quality function deployment or techniques. Typically, requirements start out abstractly, e.g., "I need a system that controls elevators." As exploration continues, they become more specific, more detailed, and less ambiguous—they split and recombine in new ways (especially when multiple cases exist). Eventually, a set of highly detailed requirements emerges, e.g., "When the 'up' button is pressed, the light behind that button illuminates within one second."

Once captured, it is extremely important to maintain traces from each requirement to its more abstract predecessor requirements and to its more detailed successor requirements. Traceability aids in change management and is a fundamental component of quality assurance and sound requirements management.

The final, most detailed requirements are contained in a document called a *requirements specification*. This specification must be communicated and agreed upon by all relevant parties. It serves as the basis for design (it tells designers what the system is supposed to do) and for test (it tells testers what the system is supposed to do). Good requirements specifications exhibit the following characteristics [1].

- Lack of ambiguity – It is unlikely your product will satisfy users' needs if a requirement has multiple interpretations.
- Completeness – Although it may be impossible to know all future requirements for a system, you should at least specify all known requirements.
- Consistency – It is impossible to build a system that satisfies all requirements if two requirements are in conflict.
- Traces to origins – The source of each requirement should be identified. It may have evolved from the refinement of a more abstract requirement, or it may have come from a specific meeting with a target user.
- Absence of design – As long as requirements address external behaviors as viewed by users or by other interfacing systems, they are still requirements regardless of their level of detail. When a requirement attempts to specify the existence of particular subcomponents or algorithms, it is no longer a requirement but rather design information.
- Enumerated requirements – Most requirements specifications enhance their readability by including auxiliary types of information that are not requirements. This information includes introductory paragraphs or sentences, summary statements, tables, and glossaries. Actual requirements contained in the document should be somehow easily discernible, whether by unique font, identifying label, or other highlighting.

A complete list of principles to adhere to when performing requirements specification appears in Chapter 3 of [2].

Writing Your SRS – Getting Off to a Good Start

Many important documents exist within your development project: descriptions of user needs, design documents, and test plans. But one particular document, the software requirements specification (SRS), is a primary concern of the software developer. The purpose of this document is to define the complete external characteristics of the system to be built. It defines all the behavioral requirements, e.g., this system shall do A when the environment does B, and nonbehavioral requirements, e.g., the system shall have an availability of 99.9 percent. Although standards are by no means a panacea, an organization that adopts a standard for the SRS achieves several benefits:

- The standard serves as a checklist of things to be addressed, so nothing is left out.
- It helps readers quickly locate and review requirements.
- It shortens the learning curve for new requirements writers and other members of the project team.

Numerous software specification standards can be used as a starting point in drafting an SRS. One that provides a good deal of guidance and flexibility is IEEE/ANSI 830-1993, IEEE Recommended Practice for Software Requirements Specifications. [3] Many other standards can be adopted to suit your needs. A good resource is the compilation by M. Dorfman and R. Thayer [4]. They have reprinted 26 different re-

---

This article derives from "Using Management Requirements to Speed Delivery of Higher-Quality Applications," Rational Software Corporation, Copyright 1995, 1996, 1997, 1998, 1999. All Rights Reserved.

quirements specifications under one cover including national, international, Institute of Electrical and Electronics Engineers (IEEE), American National Standards Institute (ANSI), NASA, and U.S. military standards.

It is important that your document outlines encourage accuracy, consistency, and a short learning curve. IEEE/ANSI 830-1993 serves as a good starting point for an SRS. Then, based on usage, you may find it beneficial to modify the standard and turn it into a corporate standard that better matches your company's specific processes and culture.

Selecting Requirements from Your Documents

Requirements documents contain some information that is not system requirements, e.g., introductions, general system descriptions, glossary of terms, and other explanatory information. Although important to an understanding of the requirements, they do not constitute requirements to be fulfilled by the system.

To ease communication of requirements and allow requirements management, writers should label those portions of text, graphics, or embedded objects that must be implemented and subsequently tested. Ideally, the requirements will be left in their original place rather than stored in multiple places; that is, they can be edited and maintained in the project documents even after they have been selected as individual requirements. This makes it easier to keep project documentation up to date as requirements change.

Organizing Your Requirements

Whether following a recognized standard or yours, you will need a section devoted to specific requirement descriptions. If you have isolated 500 requirements, for example, you should find a way to group them to aid in understanding rather than document them as a long list of bullets. We recommend organization by

- Mode of operation.
- Class of user.
- Object.
- Feature.
- Stimulus.
- Combining any of the above [1].

Applications that have clearly defined states (powered up, error recovery, etc.) could have their requirements grouped under their corresponding mode of operation. Systems that have a significant number of diverse users might be best organized by class of user. For example, a specification for an elevator control system could be organized into three major subsections: passenger, fireman, and maintenance employee. This provides a logical way to group specific requirements so that they can be reviewed and understood by each class of user.

Other applications may best be suited to organization by feature; that is, highlight the features and their intended behaviors as viewed by the user. Others, e.g., an air traffic control system, which is rich in real-world objects, may best be organized by grouping the behaviors of objects in the system. This approach may also be well suited to software organizations that have adopted object technology as their development paradigm.

Managing Requirements with Attributes

All requirements have attributes regardless of whether they are recognized. These attributes are a rich source of management information that can help you plan, communicate, and track your project's activities throughout the lifecycle. Each project has unique needs and should therefore select the attributes that are critical to its success. Following is a sample.

Customer Benefit – All requirements are not created equal. Ranking requirements by their relative importance to the end-user opens a dialogue with customers, analysts, and members of the development team.

Effort – Clearly, some requirements or changes demand more time and resources than others. Estimating the number of person-weeks or lines of code required, for example, is the best way to set expectations of what can or cannot be accomplished in a given time frame.

Development Priority – Only after considering a requirement's relative customer benefit and the effort required to implement it can the team make feature trade-offs under the twin constraints of a project's schedule and budget. Priority communicates to the entire organization which features will be done first, which will be implemented if time permits, and which will be postponed. Most projects find that categorizing the relative importance of requirements into high, medium, and low or essential, desirable, and optional is sufficient, although finer gradations are possible.

Status – Key decisions and progress should be tracked in one or more status fields. During definition of the project baseline, choices such as *proposed*, *approved*, and *incorporated* are appropriate. As you move into development, *in progress*, *implemented*, and *validated* could be used to track critical project milestones.

Authors – The names of people (or teams) responsible for the requirement should be recorded in the requirements database, whether it is the person responsible for entering the text or the person responsible for identifying the need.

Responsible Party – The person who ensures the requirement is satisfied.

Rationale – Requirements exist for specific reasons. This field records an explanation or a reference to an explanation. For example, the reference might be to a page and a line number of a product requirement specification or to a minute marker on a video tape of an important customer interview.

Date – The date a requirement was created or changed should be recorded to document its evolution.

Version of Requirement – As a requirement evolves, it is helpful to identify the version numbers (and history) of requirements changes.

Relationships to Other Requirements – There are many relationships that can be maintained between requirements. For example, attribute fields can record

- The more abstract requirement from which this requirement emanated.
- The more detailed requirement that emanated from this requirement.
- A requirement of which this requirement is a subset.

- Requirements that are subsets of this requirement.
- A requirement that must be satisfied before this requirement is satisfied.

It is especially important to maintain linkages from requirements to all development products that emanate downstream from them. By providing these links, one can easily ascertain the impact of any changes and quickly determine development status (which should be an attribute of those downstream entities).

The above list is not exhaustive. Other common attributes include stability, risk, security, safety release implemented, and functional area. Whichever method is used to track them, attributes should be easily customized to adapt to the unique needs of each team and each application.

Requirements Traceability – Ensuring Quality and Managing Change

Requirements traceability is explicitly required in most Department of Defense software contracts and is typically practiced by manufacturers of all high-reliability products and systems. In the health-care industry, requirements traceability is governed by the proposed changes in the Good Manufacturing Practices Regulation. However, most companies outside these industries do not routinely practice requirements traceability.

Traceability is a link or definable relationship between two entities. Those who use requirements traceability find that it provides a level of project control and assured quality that is difficult to achieve by any other means. At Abbott Laboratories, where traceability was instituted in 1987, they like to say, "You can't manage what you can't trace." [5] This makes intuitive sense if for no other reason than to emphasize that full requirements test coverage is virtually impossible without some form of requirements traceability.

Benefits of Requirements Tracing

In its simplest terms, requirements tracing demonstrates that software does what it is supposed to do. The key benefits of this process include

- Verification that all user needs are implemented and adequately tested.
- Verification that there are no "extra" system behaviors that cannot be traced to a user requirement.
- Understanding the impact of changing requirements.

Implementing Requirements Traceability

Figure 1 shows a sample hierarchy of project documents. In this example, the product requirements document is the "source" of all requirements. In other examples, the source document could be a user-needs document or system specification.

A hierarchical relationship between two documents in Figure 1 is an indication that interrelationships may exist among specific elements in those documents. For example, the relationship shown between the product requirements document and the software requirements specification implies that any specific product requirement could be satisfied by one or more software requirements. Similarly, any software requirement may help to satisfy one or more product requirements. Clearly, a product requirement with no related software requirements or hardware requirements will not be satisfied. The reverse also is true: A software requirement with no related product requirements is extraneous and should be eliminated.

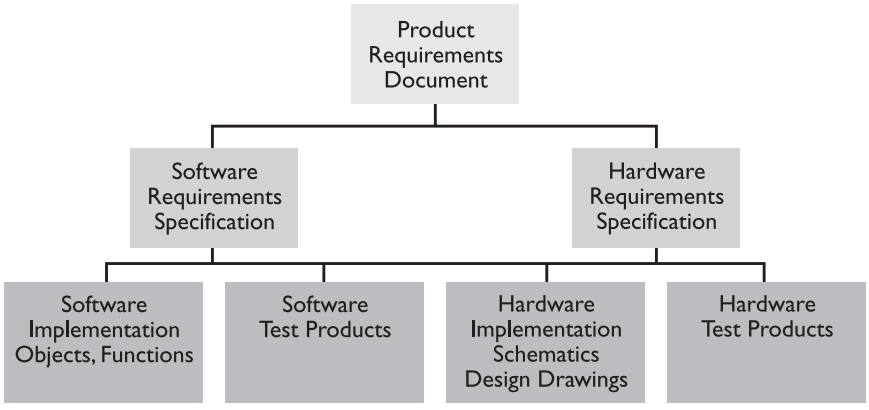
In addition to establishing document relationships to support traceability, you will need to employ some form of system to maintain links between individual items within the hierarchy. This can be done by embedding links and identifiers directly within the document or by using a separate spreadsheet or database that manages the linkages outside the document. There are advantages and disadvantages to each of these approaches. A new class of requirement management tools automatically maintains traceability links. Ideally, this capability is integrated in the same tool that manages and manipulates the documents and their individual requirements.

Change Management

Traceability provides a methodical and controlled process to manage the changes that inevitably occur as an application is developed. Without tracing, every change requires that documents be reviewed on a moment-to-moment basis to see which, if any, elements of the project require updating. Because it is difficult to establish whether all affected components have been identified, changes tend to decrease system reliability over time.

With traceability, management of a change can proceed in an orderly fashion. The impact of a change can now be understood by following the traceability relationships through the document hierarchy. For example, when a user need changes, a developer can quickly identify which software elements must be altered, a tester can pinpoint which test protocols must be revised, and managers can better determine the

Figure 1. *Example document hierarchy.*



What You Can Do

- Continue to educate yourself on the benefits of requirements management. Secure training. Read. We have included a suggested list at the end of this article.
- Explore and use the new tools that make requirements management easier.
- Adopt a personal strategy to better communicate the requirements you own.

potential costs and difficulty to implement the change.

Requirements Reporting – Easing Management Reviews

A requirements repository gives managers a powerful tool to track and report project status. Critical milestones are more easily identified. Schedule risks are better quantified. Priorities and ownership are kept visible. Querying the repository can quickly uncover facts that provide answers to important questions, such as

- How many requirements are there on this project? How many are high priority?
- What percentage of the requirements are incorporated in the baseline?
- When will they be implemented?
- Which requirements changed since the last customer review?
- Who is responsible for the changes?
- What is the estimated cost impact of the proposed changes?

High-level reports aid management reviews of product features. Requirements can be prioritized by user safety considerations or by customer need, difficulty, and cost to implement. These specialized reports help managers better allocate scarce resources by focusing attention on key project issues. The net result is that managers make better decisions and thereby improve the outcomes of their company's application development efforts.

Conclusion

Software development is one of the most exciting and rewarding careers of our time. Unfortunately, many of us carry the scars from applications that missed expectations. It is common for applications to overshoot their schedule by half, deliver less than originally promised, or be canceled before release. To keep pace with rising complexity and increased user demands, we must begin to mature the ways in which we develop, test, and manage our software projects. The first step in this advancement is improved requirements management.

Requirements management provides a "live" repository of application requirements and their associated attributes and linkages. This repository establishes an agreement on exactly what the software is supposed to do. It provides a wealth of information that can be used to manage and control your projects. Your quality will improve, and the software you build will better fit your customer's needs. And with requirements data available to all members, team communication is greatly improved.

Start now. You can cut project costs significantly by catching requirement errors early. Try to write down, in plain English, all the requirements of your current project. (Hint: if this is difficult or was not already done, ask why.) Compile these requirements in a suitable, short report and share them with your customers and peers. Get their feedback. Are any requirements missing, incomplete, or wrong? There is a good chance the answer is yes to all three. If it is still early enough to correct these errors, you have saved a lot of money. If it is too late, ask what is it about your process that could change, then propose a first step. ◆

About the Authors

Alan M. Davis is founder and chief executive officer of Omni-Vista, Inc., which develops and markets software development decision-making support tools. He serves as professor of Computer Science and El Pomar professor of software engineering at the University of Colorado at Colorado Springs. He is author of *Software Requirements: Objects, Functions, and States* and *201 Principles of Software Engineering* and is author or co-author of more than 100 papers on software and requirements engineering. He was editor in chief of *IEEE Software Magazine* from 1994 to 1998.

Dean A. Leffingwell is a vice president of Rational Software and is general manager of Rational University, where he is responsible for methodology, the Rational Unified Process, and customer education and training. Before 1997, he was chief executive officer and co-founder of Requisite, Inc., developers of the RequisitePro requirements management product and Requirements College™. He is considered an authority in requirements management and software quality and is a frequent speaker on these topics. He has a master's degree in engineering from the University of Colorado.

References

1. *CHAOS*, The Standish Group International, Inc., Dennis, Mass., 1994.
2. Davis, A., *201 Principles of Software Development*, McGraw-Hill, New York, 1995.
3. http://standards.ieee.org/catalog/olis/arch_swe.html
4. Dorfman, M. and R. Thayer, *Standards, Guidelines and Examples of System and Software Requirements Engineering*, IEEE Computer Society, Los Alamitos, Calif., 1991.
5. Watkins, R. and M. Neal, "Why and How of Requirements Tracing," *IEEE Software*, July 1994, pp. 104-106.

Suggested Reading

1. Davis, Alan M., *Software Requirements – Objects, Functions, and States*, Prentice-Hall, Englewood Cliffs, N.J., 1993.
2. Gause, Donald C. and G. Weinberg, *Exploring Requirements – Quality Before Design*, Dorset House, New York, 1989.

For information on how to order these books or for addresses of the available Internet forums that discuss requirements management, please write, call, or send E-mail to

Rational Software Corporation
18880 Homestead Road
Cupertino, CA 95014
Voice: 800-728-1212
Fax: 408-863-4120
E-mail: info@rational.com
Internet: http://www.rational.com.

Sponsoring Process Improvement

Col. Henry M. Mason
U.S. Air Force

Institutionalizing a process for continuous improvement in an organization requires active sponsorship from a leader. It can be particularly challenging for a leader to sustain momentum for long-term process improvement in an organization that excels at crisis management. This article presents lessons learned from sponsorship of a process improvement effort based on the Software Acquisition Capability Maturity Model (SA-CMM℠) at Warner Robins Air Logistics Center (WR-ALC) Special Operations Forces Directorate (LU).

In 1997, WR-ALC/LU initiated an internal process improvement effort based on the Software Engineering Institute (SEI) SA-CMM. Our goals were to institute a process for continuous process improvement, to become a knowledgeable and efficient acquisition organization, and to achieve SA-CMM Level 2 within 18 months. The aim of our improvement effort, which we called ASPIRE (Acquisition and Sustainment Process Improvement/Re-engineering Effort), was to improve the software acquisition and sustainment processes of the System Program Office (SPO), including the ability to

- Acquire and deliver systems in less time.
- Reduce development costs.
- Reduce lifecycle costs.
- Deliver highly reliable software-intensive systems that meet the needs of our customers.

This article shares what the directorate has learned from this effort to date, which is, that process improvement is harder than it may appear.

Process Improvement

On June 27, 1997, LU completed an SA-CMM-based assessment for internal process improvement and finalized the results in a Findings and Recommendations Report prepared in August 1997. With the help of the SEI and the Software Technology Support Center (STSC), we clarified the roles and responsibilities of our Management Steering Group (MSG) to provide management and direction and our System Process Improvement Network (SPIN) team to oversee implementation of technical process improvement. Initially, six Process Action Teams (PATs) of five to six members were established to define and develop software acquisition processes, and additional PATs will be established as we proceed.

The obvious reason we chose to apply the SA-CMM was to improve the directorate's expertise in software acquisition. However, software acquisition was of interest to only a small number of people within the SPO. Because some acquisition processes are common to both software and hardware, we hoped that the improvements that would lead us to achieve SA-CMM Level 2 would help us improve acquisition in general. Therefore, we used the SA-CMM to learn how to institute process improvement and applied the techniques of SA-CMM-based process improvement to the larger organizational context.

My experiences with improvement programs (Total Quality Management [TQM], Zero Defects, and Management by Objectives) had shown me how difficult it is to institutionalize a process for continuous improvement. In our organization, as in many others, process improvement tends to follow a 24-month fad cycle; when the cycle is completed, lasting improvements can be difficult to identify. Nevertheless, the discipline of the SA-CMM model, along with the available expertise from the SEI and the STSC, offered the hope that the SA-CMM could be used to achieve a higher purpose: an institutionalized process by which things get better in the SPO.

I expected that institutionalization of process improvement would take about three years. I would be at WR-ALC/LU for three years, so I thought we had a chance to make a good start. My goal as a sponsor was to make process improvement a significant enough part of daily processes and create sufficient momentum so the SPO would have a reasonable chance to sustain process improvement beyond my tenure.

To reinforce the message that process improvement should be a normal part of its everyday work, the organization has been extremely careful not to make process improvement a program. Reengineering was a program; TQM was a program, and I did not want to make the "program" mistake again. While at the Air Power Institute, I wrote a book on TQM and how it could be used on the flight line [1]. At that time, TQM literature was not extensive. I found that when TQM failed—which it most often did—it was because it gave people tools before it identified the problems the tools were intended to solve. As a result, people used these tools to fix annoyances: "Let's get rid of staff meetings and repave the parking lot." With its focus on improvement of the software acquisition process, the SA-CMM seemed to be a way to keep the implementation of TQM grounded in real, immediately pressing problems.

Allocating and Committing Resources: Process Improvement in the Context of Crisis Management

There is an old saying, "If you always do what you did, you always get what you got." It can be particularly difficult to change old habits in an organization in which everything is crisis management. As a Special Operations Forces (SOF) organization, WR-ALC/LU is, by necessity, highly skilled at crisis management.

CMM is registered in the U.S. Patent and Trademark Office.

In the language of the SEI CMMs, we were a typical Level 1 organization—ours was a culture of institutionalized heroism. But in a culture of heroism, nothing gets better, heroes retire, and their skills retire with them. Ironically, we found that our skill at crisis management was a liability when it came to instituting process improvement. Employees had a tendency to say, "I don't have time for this quality stuff; I have a job to do."

In light of this tendency, we established a rule that *never* would more than 5 percent of our total SPO resources be devoted to process improvement. This rule gave me a powerful way to combat resistance. Whenever someone complained about the overhead that process improvement would add, I would say, "Surely you can do your work with 95 percent of your resources." In practice, we have never used more than 4 percent of our resources on process improvement, and we averaged around 2.5 percent. I did not take much of their resources. If I had pushed employees harder—to dedicate around 5 percent to 6 percent—I would have overtaxed them. On the other hand, an effort of about 1 percent would not be enough to sustain improvement. At the 2.5 percent level, I knew that we would make steady, measured progress without burning out. I also knew that the effort would not fizzle out and die.

Metrics and Process Improvement

Metrics are important to project success; however, a manager who manages only with metrics is probably easy to deceive. I have discovered that things that are easy to measure are often not particularly important, and things that are the most difficult to measure tend to be the most important. For example, I check the schedules of the PATs against their progress, and I listen to everybody in the MSG meetings to gauge attitudes about how well senior staff is integrating process improvement into how we do business. I believe attitudes are probably the most important barometer of success. A metric I track carefully is the amount of effort we spend on process improvement across the SPO. This metric tells me if the effort is increasing, decreasing, or staying about the same.

To reinforce the idea that we could improve our processes and not place too much strain on our resources, I had to demonstrate my willingness as a leader to apply the resources that I controlled to the effort. For example, to convey that the MSG was not add-on work, I released employees from private staff meetings to participate in the MSG. Setting a bound on resource commitments sent the message that process improvement is not a periodic, overwhelming demand on the employees' time that has a beginning and an end but is the normal way we do business. The MSG is now perceived as a part of everyday work processes.

Sponsorship

Process improvement efforts are a constant test of senior leadership. You have to back up your talk with actions and you cannot waver. If leadership wavers and process improvement moves down the scale of importance, the effort will die. Additionally, if sponsors establish and reinforce a vision for the effort, they move beyond passive endorsement to active sponsorship.

Although we faced challenges, such as increased competition and a potential loss of market share, we did not have a significant emotional event to trigger process improvement changes. When your livelihood and your life do not depend on change, it is an uphill battle to sustain commitment for process improvement; therefore, it was essential to remind everyone at least every six months, via correcting meetings called "visioning" sessions, where we were going and why we were going there.

To sustain commitment is most difficult in the early stages of the effort, before there are tangible results you can touch, feel, or sense. Therefore, leadership must strike a balance between patience and active engagement.

If you are the type of leader who relies only on the evidence available to your senses, you will fail. In the beginning, everything is intuitive and conceptual, and leaders must be willing to let the process percolate and allow employees to find their own solutions to the problems they encounter. I adapted our processes to this new way of operating. If I had pushed too hard in the early days of our effort, we may have achieved some ephemeral success, written it up, congratulated ourselves, and terminated the program. I believe patience is the most essential quality for the leader of a process improvement effort. If you do not have patience, it is best not to go through the pain. Abort early and avoid the rush.

On the other hand, the leader must also know when to push. If anything, I probably erred on the side of patience. Attempting to see if the effort could sustain itself with minimal intervention in the MSG, I stepped back too far too soon. For four or five months in the second year of the effort, I was not actively engaged in the process. Eventually, the SPIN let me know that the effort needed my active participation.

Visioning

In March 1998, at a meeting of the MSG, we held our first visioning session. We formulated a vision of an improved organization, a picture of what it would be like to work in that organization, and a list of the expected payoffs. This strategy required the MSG to be directly involved in the improvement process. At this meeting, I addressed the staff: "I'm willing to quit right now. If you don't want to do this, we'll stop. You know why I think it's important. I'm not going to be here forever. We don't need to drag this out until I retire 18 months from now. If you continue, things will get better, and you'll see many of the benefits I've been preaching about. But we don't have to do this. I am willing to disband the SPIN team and cancel all the PATs, and we'll go back to what we were doing. If you see long-term benefits for the SPO and for yourselves, then I want you to make the commitment. But you will have to agree to support the PATs. The decision is yours." And I left it to them to decide. This meeting became a watershed when the rest of the MSG, without my influence and after much discussion, decided to continue the improvement effort.

Prior to this meeting, the PATs had not produced anything, and enthusiasm

The Fifth Annual Joint Aerospace Weapon Systems Support, Sensors, and Simulation Symposium and Exhibition

"Making Information Work for the Warfighter"

June 13 – 17, 1999
Radisson Hotel, Mission Valley
San Diego, Calif.

Sponsored by: U.S. Air Force Embedded Computer Resources Support Improvement Program, U.S. Air Force, U.S. Army, U.S. Navy, and U.S. Marine Corps.

Managed by: U.S. Air Force Software Technology Support Center and the National Defense Industrial Association

The symposium will also include Briefing to Industry presentations and a classified session.

Visit our Web site for current program information or call conference managers for more information.

This will be a great conference at a great place at a great time! See everyone there!

Dana Dovenbarger, Conference Manager
Lynne Wade, Assistant Conference Manager
Software Technology Support Center
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: 801-777-7411 DSN 777-7411
Fax: 801-775-4932 DSN 775-4932
E-mail: dovenbad@software.hill.af.mil
Wadel@software.hill.af.mil
Internet: http://www.jawswg.hill.af.mil

for the effort began to wane. We had found lots of ways to do nothing. The PATs realized that a group that works for only two hours every week will get little done over a long period of time. Eventually, we implemented an approach where employees would block out two weeks and cram; however, you must plan for and schedule these concentrated meetings at least three months in advance because most employees' calendars are full in the near term. PAT and SPIN productivity was the result of teams maturing as they went deeper into process development.

Since the visioning session and the changes initiated by the SPIN and the PATs, I have noticed much more enthusiasm from employees. PATs have begun to complete their work, and the results have been encouraging. For example, the first PAT, Acquisition Life Cycle Checklist (ALCC), dealt with the SA-CMM in general terms. We developed a checklist of every action necessary to add new capability or to enhance an existing capability on SOF weapons systems (concept development through system life sustainment). The checklist applied to all disciplines in the SPO. For the first time, we had a comprehensive layout of this extremely complex process. We used the ALCC to develop program management plans and schedules, as an on-the-job training tool, and as a management tool to track program progression from development through system installation. The checklist and associated training were well received by the work force; one software engineer with 15 years experience commented that she wished she had the checklist 15 years ago.

Our second PAT, Software Fielding Process, dealt with an acute LU problem. Acquisition reform and base realignments had removed the infrastructure that supported new software distribution to the war fighter. This PAT developed a process to immediately disseminate software through a password-protected, secure Web site. User organizations have successfully tested the system and are excited about the immediate accessibility they now have. With this new process, software changes can be available to user organizations within hours of software acceptance.

Another sign of growing acceptance is that those who participated in the early PATs have volunteered to join new PATs. Our ongoing PATs address risk management, standardized cost estimation, training, and solicitation policy and planning. At the staff's request, I did not attend our "revisioning" session held in October 1998. As they wrestled with recommitment to ASPIRE, as well as meeting the demands of day-to-day challenges, the staff wanted the freedom to air their problems, differences, and gripes and develop their plan to help me manage the organization. The results were especially satisfying. Each senior manager accepted the challenge to wholeheartedly support ASPIRE and manage the SPO business as a unified group, now identified as the Management Working Group (MWG). The MWG meets monthly without my direct intervention, and the MSG meetings are now quarterly sessions. To keep our direction on track, I provide coaching and steering that is in line with our SOF SPO mission and goals. At our next revisioning session, we will check our progress by identifying what has and has not been working and what we need to change in our approach.

I have no doubt that there will be another period after the early successes have been achieved and instituted when everyone says, "Okay, now we are done." The idea has not yet fully sunk in that we have a system for process improvement. If something is wrong with our work processes, we can feed the problem into the new system and allow the system to take care of it and come out with a new process that is implemented, reviewed, and updated. Our organization will not have sufficient confidence in our processes until we realize that process improvement is forever.

Lessons Learned

To summarize, following are some key lessons I learned as sponsor of our process improvement effort.

- Do not characterize process improvement as a separate "program"; characterize it as the normal way of doing business.
- Spend no more than 5 percent of total organizational resources on the improvement effort.
- To monitor progress of the effort, track resources spent on improvement, track PAT progress against schedules, and pay attention to attitude changes.
- Whenever possible, demonstrate your commitment by applying resources that you control to the effort.
- Clearly identify and communicate the problems that process improvement are intended to solve.
- Establish a vision for the effort, and at least every six months, reinforce the vision and the commitment to achieving it.
- Enable PATs to meet for concentrated periods; schedule the meeting times several months in advance.
- Find the right balance between patience and active engagement. ◆

Acknowledgment

I thank Bill Pollak of the SEI for his help in preparing this article for publication.

About the Author

Col. Henry M. Mason is the director of SOF SPO at Warner Robins Air Logistics Center, Robins Air Force Base, Ga. He is the single authority for fleet management for the U.S. Special Operations Command and Headquarters Air Force Special Operations Command for fixed-wing aircraft. He directs more than 420 personnel at Robins Air Force Base and at Wright-Patterson Air Force Base, Ohio, and he directs more than 20 contractor support agencies.

Mason has a bachelor's degree in engineering management from the U.S. Air force Academy, a master's degree in business administration from the University of California at Los Angeles, and is a graduate of the Air War College, Maxwell Air Force Base, Ala. He is a command pilot with more than 2,000 flying hours, and he has commanded at the squadron and group level. His duty assignments include operational flying, flight safety, aircraft maintenance, logistics, and acquisition program management.

Point of Contact
Chuck Idone
226 Cochran Street
Robins AFB, GA 31098
Voice: 912-926-6078
E-mail: cidone@lu.robins.af.mil

Reference

1. "Quality Flight-Line Maintenance," Air Power Research Institute, U.S. Air Force, Maxwell Air Force Base, Ala., 1989.

Coming Events

Eleventh Annual Software Technology Conference

Dates: May 2-6, 1999
Location: Salt Palace Convention Center, Salt Lake City, Utah
Co-sponsors: U.S. Air Force, U.S. Army, U.S. Navy, U.S. Marine Corps, Defense Information Systems Agency, and Utah State University Extension
Co-hosts: Ogden Air Logistics Center and the Software Technology Support Center
Theme: "Software and Systems for the Next Millennium"
Contact: Dana Dovenbarger or Lynne Wade
Voice: 801-777-7411 DSN 777-7411
Fax: 801-775-4932 DSN 775-4932
E-mail: dovenbad@software.hill.af.mil or wadel@software.hill.af.mil
Internet: http://www.stc-online.org

Project Management for Development of Software-Intensive Systems

Dates: May 10-12, 1999
Location: University of California at Los Angeles (UCLA)
Subject: This course presents practical techniques and tools to estimate, plan, lead, organize, control, and complete high-quality projects that are within budget and on schedule and that meet the needs of the customer.
Sponsor: UCLA Extension Short Course Program
Contact: Donald S. Remer
Voice: 909-621-8964
E-mail: remer@hmc.edu
Contact: Marcus Hennessy
Voice: 310-825-1047
E-mail: mhenness@unex.ucla.edu
Internet: http://www.unex.ucla.edu/shortcourses

Montgomery Area Golf Outing and Information Technology Partnership Day 1999

Dates: May 17-18, 1999
Locations: Wynlakes Golf and Country Club and Embassy Suites Hotel, Montgomery, Ala.
Theme: "Y2K and Challenges Beyond – Government and Industry Shared Solutions"
Chairman: William R. Stevenson
Voice: 334-416-4041
Fax: 334-416-5505
E-mail: william.stevenson@gunter.af.mil
Internet: http://web1.ssg.gunter.af.mil/partnership

Gaining Confidence in Using Return on Investment and Earned Value

Larry W. Smith, *Software Technology Support Center*
A. Todd Steadman, *TRW Avionics Systems Division*

The terms "earned value" and "return on investment" (ROI) are frequently heard in the world of project management; however, they are often used incorrectly or inconsistently. This article summarizes the major principles and purposes of these management tools with the intent on moving them into the mainstream of proper use.

Throughout the Department of Defense (DoD) and private industry, the terms "return on investment" and "earned value" are becoming more commonplace. Their use also is becoming more appropriate, visibly demonstrated, and validated. These terms tend to frustrate some program managers and corporate executives, while other organizations revel in their daily application. It is interesting that some organizations value these terms, while others disregard or minimize their use. At the heart of this conundrum lie a variety of statements and questions that range from skepticism to downright confusion:

- Is the definition believable? "This project showed an ROI of 1,421 percent over three and a half years with a payback period of 0.23 years."
- Are you sure? "With respect to project status, I believe we are right on course and are actually underrunning our costs."
- Lack of understanding. "Considering project's nature, what is the best method to compute our ROI?"
- Complexity and confusion. "How in the world can I compute earned value on *that* task?"

In working with a variety of organizations, the Software Technology Support Center (STSC) has discovered the equal variety of approaches to ROI and earned value. Some organizations tend to only intellectually capture and display the information, others use inappropriate or incorrect definitions to support marketing tactics, while others honestly struggle with accurate definitions but value these tools as legitimate management approaches.

As an example, the STSC recently provided support to a DoD organization in the inspection of a software development plan. The inspection was carefully planned, then implemented over a short time frame. At the outset, ROI and earned value were not a specific focus of the effort, and detailed measures were not defined. However, as the inspection neared completion, a sufficient amount of data was generated that enabled the organization to compute rough estimates for both earned value and ROI.

By deliberate design, the partitioning (chunking) of the document, the allocations of inspection assignments, and the regular monitoring of progress enabled a clear estimate of the earned value achieved at different points during the inspection. Once the data was baselined, earned value was a straightforward computation that demonstrated a progression toward successful completion of the inspection project.

The organization also was pleased with a relatively accurate ROI estimate of just over a ratio of 4-to-1. This ROI measure was based on actual hours invested in inspecting the document and estimated hours saved in downstream costs if the discovered defects had not been detected. This estimate worked well for this activity and measurably justified the expenditure of effort. However, since it was unique to this organization and this effort, mapping it to another situation would be inappropriate. Although not specifically quantifiable, the ROI was influenced by strong leadership, an insistence on progress tracking, and the commitment of the inspection team—all semiunique intangibles.

Earned value and ROI have been called management indicators, metrics, measurements, etc. Although ROI is typically used as an overall indicator of project success, it is supported by the consistent tracking and monitoring of one of the "smaller scale" (but more quantifiable) ROI measures—earned value. Conscious and regular earned-value measurements point toward and validate any stated ROI. Therefore, the proven utility and importance of these two tools—used not only separately but also together—are reasons to understand them better. The following sections define the terms ROI and earned value, establish a context for their use, and discuss a few examples of how they might be applied.

Return on Investment

In its basic computation, ROI is stated as the *ratio of savings estimated or measured in a given effort by the cost incurred to accomplish that effort*. The difficulty in computing ROI is to determine what constitutes the *total savings* or *return* and what constitutes the *total investment*. Although much of the value that goes into the terms *total return* and *total investment* may be straightforward and measurable, these terms often include various intangibles that are not only valid but also crucial to an accurate measurement of ROI. Less tangible items could include customer confidence, competitiveness, effects of downtime, impact on productivity, and lost opportunities.

Common Terminology

A discussion of the terminology associated with ROI and earned value will assist in understanding their usefulness and fostering practical implementation. Note that the ROI terms and associated definitions were compiled in part from [1]; earned-value terms and associated definitions were compiled in part from [2] and [3]. See Figure 3 for a graphical representation of some of these terms.

ROI Terms

Payback Period – The amount of time following a project or improvement effort, either estimated or measured, during which the total investment of the improvement will be repaid by the savings it brings.

Investment – The estimated or measured total cost in hours, dollars, or other units that an improvement effort requires to be planned, executed, and completed.

Return on Investment – The total quantitative savings or return, in hours, dollars, or other measurable units, generated by an improvement effort, divided by the total cost of the improvement effort.

Cost of Quality – A popular factor in the computation of ROI for quality-related improvement efforts. Cost of quality is the cost of not doing things right the first time, and may include preparation costs, execution costs, and follow-up costs for the effort as well as other measures that contribute to the quality effort.

Cost of Conformance – The estimated or measured cost for an organization or project to conform to stated requirements. Cost of conformance generally includes assessment costs and prevention cost.

Cost of Nonconformance – The estimated or measured cost incurred by an organization or project for reworking an effort or project because things were not done correctly the first time.

Earned-Value Terms

Budgeted cost of work scheduled (BCWS) – The sum of the budgets for all planned work scheduled to be accomplished within a given period.

Budgeted cost of work performed (BCWP) – Also called the earned value, it has three definitions: (1) The estimated (in contrast to the planned) value of work performed as of a specific point in time, (2) a method for measuring project performance comparing the amount of work that was planned with what was actually accomplished to determine if cost and schedule performance is as planned [2], and (3) the sum of the budgets for completed work and the completed portions of open work [3].

Actual cost of work performed (ACWP) – The costs incurred in accomplishing the work performed.

Schedule variance (SV) – The numerical difference between the budgeted cost of work performed and the budgeted cost of work scheduled.

Cost variance (CV) – The numerical difference between the budgeted cost of work performed and the actual cost of work performed.

Schedule performance indicator (SPI) – The planned schedule efficiency factor representing the relationship between the value of the initial planned schedule and the value of the physical work performed.

Cost performance indicator (CPI) – The cost efficiency factor representing the relationship between the actual costs expended and the value of the physical work performed.

Budget at completion (BAC) – The sum of all budgets allocated to a project. The BAC is synonymous with the performance measurement baseline.

Performance measurement baseline (PMB) – The time-phased budget plan against which project performance is measured. The PMB is synonymous with the BAC.

When such complexities are factored in, the ROI computation usually changes, considering diverse elements inherent in the improvement effort it is meant to quantify. Therefore, because it is less costly to compute an estimated ROI (because of the intangible values), one could reasonably conclude that most stated ROIs are heavily estimated rather than measured. This causes difficulty in comparing ROI information.

To better understand the ROI concept, examine different uses and approaches to compute and report ROI. Any organization that desires to successfully deliver products or services over a long term requires a positive ROI. It follows that good management planning will outline steps necessary to reach such an ROI. This includes selecting the proper method to compute ROI, accurately reporting ROI, identifying necessary success factors that relate directly to your product and organizational structure, then strategically leveraging them.

Improvement-Based ROI – Depending on the type of improvement being attempted, approaches to compute ROI may be drastically different. The end product of one type of improvement may yield highly quantifiable, or at least estimable, results. For example, research of the potential benefits of document inspections has led to estimated hourly savings to find and fix defects upstream in the lifecycle as opposed to finding and fixing them in later development phases [4, 5]. The estimated savings per defect, although questioned by some, translate directly to personnel hours and dollars saved by the inspection process improvement, due to earlier and more efficient detection and removal of defects.

Conversely, the insertion of a network management system, for example, may not yield an easily quantifiable ROI. Although there would be some readily quantifiable costs (design, development, operation, acquisition, training, and maintenance), many critical costs and benefits may be far less tangible, as discussed earlier.

In such cases, the difficult task for managers is to quantify these intangibles. Although difficult, the task is not impossible. One often overlooked method is the strategic use of customer satisfaction surveys [6]. For example,

frequently surveying key customers might reveal that when the overall satisfaction level exceeds a given threshold, customers are a certain percentage more likely to request additional support, add task orders to contracts, and rule favorably on contract incentives. Similar information gathering at the marketing level may reveal that an estimated dollar value of contracts were won due in part to the improvement effort and its effect on the organization's capabilities and performance. Understanding what the intangibles are and how they affect the organization is key to understanding how to compute ROI for them.

Reporting ROI – In many cases, the method of reporting ROI is as much a key to success as the computation of ROI. For this reason, organizations often report ROI measurements along with additional reference points, such as payback period, risk mitigation strategies, and forecasted ROI estimates. Upper management is continually interested in the "quick win" or "low-hanging fruit" in an improvement effort. Factual and substantiated estimates that reinforce a stated ROI value provide that benefit to management—benefits that go a long way toward continued sponsorship of the effort, professional credibility, and willingness to accept risk and continue in improvement initiatives.

As members of an organization become better at the computation and management of ROI, they become more adept in computing the amount of time in which an improvement will pay for itself. The organization will be more able to identify risks to the effort and propose mitigation that will preserve and even increase the ROI. In this optimized environment, the organization will be able to estimate with considerable accuracy the ROI expected in the coming months and years as the improvement effort progresses.

ROI Success Factors – Organizations that are likely to achieve the best ROI are those that embrace it as a strategy [6]. The natural product of any strategy is a plan or road map that, if followed, will achieve some measure of success. The following elements will contribute to such a plan.

- Use the principles of cost-and-benefit analysis to completely identify all potential costs and savings expected as a result of the improvement effort.
- Rank all costs and savings of the effort with respect to the severity or importance of their impact.
- Identify the subset of costs and savings that will be used to compute the ROI.
- Develop a plan to monitor and manage the selected costs and savings.
- When computing ROI, plan for the inclusion of input from people who have a direct understanding of the non-tangible benefits or savings, e.g., marketing, engineering, or contracting departments.
- Frequently communicate about the plan with all relevant parties.
- Compute and use additional supporting metrics such as payback period.
- Determine strategies to improve the estimated and forecasted ROI for the coming months or years.
- Strengthen sponsorship by including key people in ROI plans and management activities.

Earned Value

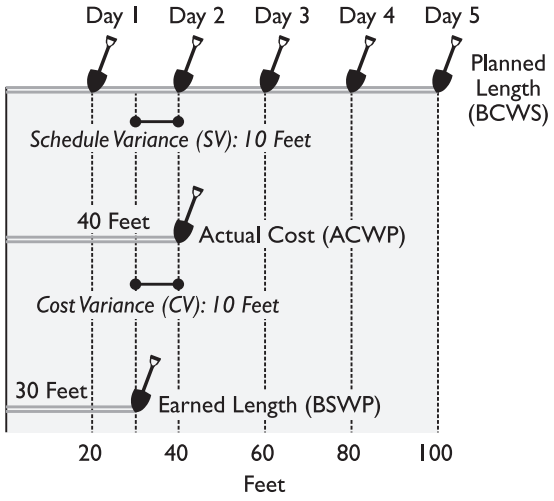
The earned value approach to project tracking originated over 100 years ago as a result of improvement efforts in the operation of factories and has gained considerable popularity in the last few decades. Earned value was formally proposed approximately 30 years ago and was implemented as a pilot project in the Minuteman missile program. Success in that effort prompted what has become known today in the DoD as the Cost/Schedule Control Systems Criteria (C/SCSC) and in private industry as the Earned Value Management System (EVMS). The C/SCSC currently consists of 32 management criteria detailed in the DoD's acquisition policy document *DoD Instruction 5000.2R.* Refer to [7] for an excellent treatment of the management control factors that originally made up the C/SCSC.

Before earned value, the traditional approach to cost and funding management was based on a project expenditure plan. This plan identified a specific funding expense rate over the duration of the project. As progress on the project was made, the total cost expended on the project to date was compared to the planned funding rate. This comparison enabled management to determine whether expenditures were ahead of or behind the amount planned for the project at that time.

The current approach of earned value adds a third dimension to this process: A quantitative estimate is made of the *value* of the work performed. This estimate represents a measure of "what you got for what you paid." Comparing the earned value to the planned cost for a given period identifies whether the project is ahead of or behind schedule, also called schedule variance (SV). Likewise, comparing the earned value to the costs expended during the period identifies whether the project is underrunning or overrunning its budget, also called cost variance (CV).

Figure 1 illustrates the basic, but often nonintuitive, principles of earned value. Assume you have contracted with an excavator to dig a 100-foot ditch over the next five days. You carefully plan the effort, deciding that total ditch

Figure 1. *The earned-value approach to project tracking.*



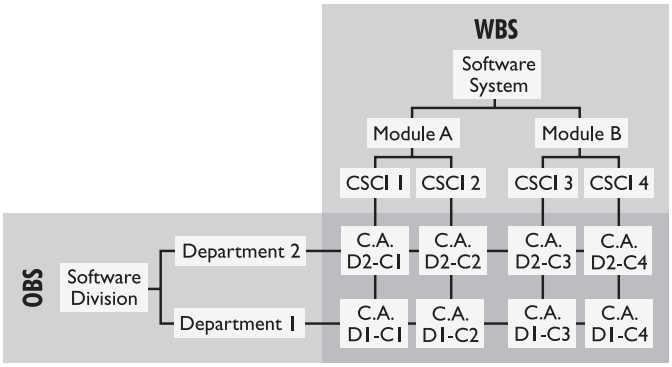


Figure 2. *Diagram showing the creation of earned-value cost accounts as the intersection of the organizational breakdown structure and the work breakdown structure.*

length dug will be the primary measurement. You scientifically calculate that 20 feet will be dug each day at an agreed hourly wage. The project begins, and after the first day, a 20-foot length of ditch is done. You measure progress and see that according to your plan, you are on schedule and within budget.

The next day is different. Because of unforeseen delays (equipment malfunction, zoning problems, volcanic intrusion, etc.), only 10 more feet are completed. Measuring progress, you find that the "earned value" of your trench is now a total of 30 feet. However, your planned value is 40 feet (two days at 20 feet per day). Furthermore, your actual ditch cost is based on two full days of digging. Therefore, the difference between your earned value of 30 feet and the planned value of 40 feet indicates that after two days, the ditch is 10 feet behind schedule or 10 percent of the planned total ditch length.

Similarly, the difference between the earned value cost of the ditch and the actual ditch cost indicates a potential cost overrun. Additional costs (replacement equipment, zoning fees, explosives, etc.) will simply add to the actual ditch cost, increasing the cost overrun. Assuming no further delays and no acceleration of the digging, 10 more feet of ditch will have to be dug at the end of the five days, adding to the total cost of the ditch.

The measurements of budgeted costs, earned value, and actual costs are generally expressed in dollars or hours. Performance estimates of earned value may be based on lines of code developed, functional units completed, etc., which are then converted to the appropriate units. Earned value provides the cost and expenditure forecast capabilities of traditional cost and funding management but adds the crucial capability of schedule estimation that the traditional approach lacks.

The biggest challenge the earned-value approach has faced has been its association with C/SCSC. C/SCSC has a proven track record as a means to manage and control large projects. From a general perspective, projects for which C/SCSC is both appropriate and usually mandated constitute only 1 percent of all projects [3]. However, the 32 criteria contained in the current version of C/SCSC are considered cumbersome and likely overkill for the remaining 99 percent.

Still, the principles behind the earned-value approach are both applicable and appropriate for these projects. These principles include proper use of work breakdown structures, cost accounts, performance measurement baselines, selection of appropriate methods to compute earned value, forecasting project performance, and capitalizing upon proven success factors.

The Work Breakdown Structure (WBS) – The use of an appropriate WBS is at the heart of C/SCSC and earned value. All work defined and subsequently tracked by the project can be located within the structure of the WBS. C/SCSC projects usually consist of a two-part WBS. The first two or three levels constitute the *contract work breakdown structure* (CWBS). The CWBS is often defined by the project owner and shows the way that cost and schedule will be monitored and reported throughout the project lifecycle. The project managers and technical team members define the subsequent levels constituting the *project work breakdown structure* (PWBS). At the lowest level of the PWBS, the tasks can be traced directly to project deliverables called out in the project's technical statement of work. At the lowest levels is the primary tracking mechanism of C/SCSC and earned value: the cost account.

The Cost Account – Cost accounts are created at the intersection of the organizational breakdown structure and the PWBS (see Figure 2). The resulting intersection creates a performance measurement unit that combines the schedule, cost, and technical aspects of the project. C/SCSC defines the cost account as, "A management control point at which actual costs may be accumulated and compared to the budgeted cost of work performed. A cost account is a natural control point for cost and schedule planning and control, because it represents the work assigned to one responsible organizational element on one contract work breakdown structure (CWBS) element." [8]

Cost accounts provide a correlation between the amount of work that is planned and the resources available to accomplish that work. Each cost account generally contains three pieces of information: the scope of work for the associated WBS element, its schedule, and its budgeted cost.

Performance Measurement Baseline – When the collection of cost accounts are summarized upward, the entire project scope, schedule, and planned cost can be determined. In C/SCSC and earned value, this information is called the "performance measurement baseline" (PMB). C/SCSC defines the PMB as, "The time-phased budget plan against which contract performance is measured. It is formed by the budgets assigned to scheduled cost accounts and the applicable indirect budgets. … It equals the total allocated budget, less management reserve." Using the PMB at any time during the performance of the project, the PMB allows the project manager to compare tracking information. Comparing the estimated earned value with the PMB at a given time yields the schedule variance for the project. Similarly, comparing the

Measurement Method	Description	Notes
Weighted Milestone	Useful for short-span tasks. Weighted budget amounts are applied to milestones distributed across the duration of the task. As the milestone is reached, the budget amount is earned.	
Fixed Formula (0/100; 25/75; 50/50)	Useful for detailed short-span tasks. An amount ranging from 0 to 100 percent of the task's budget is earned when the task begins. The remaining percentage is earned when the task is complete.	These measures are typically used to track earned value on non-recurring tasks.
Percent Complete Estimates	Easiest method to administer. A "subjective" estimate of the percentage of work completed is used for the earned value. Requires well-defined work packages and guidelines to determine an accurate percent-complete value.	
Percent Complete and Milestone Gates	Popular method used by government organizations. Uses a combination of weighted milestones and percent complete. "Subjective" percent-complete estimates are allowed up to a specific ceiling associated with each milestone. Advancement past the milestone is not allowed until tangible criteria have been met, hence the term "gate."	
Equivalent Completed Units	Useful for extended duration and repetitive tasks. The overall project is divided into distinct units of accomplishment. Earned value is computed by summing the units completed.	
Earned Standards	Useful for production-type work. Perhaps the most sophisticated method to compute earned value. Standards of performance based on historical cost data, time and motion studies, etc., are used to compute the earned value on a given task. Often, several standards are used simultaneously, and a management consensus determines which standard is ultimately used.	These measures are typically used to track earned value on either nonrecurring or recurring tasks.
Apportioned Relationship to Discrete Work	Useful for tasks in which their performance has a direct relationship to other tasks. The earned value for apportioned tasks is a summary of the earned value measurements made on the work to which it is related. Schedule variances in the apportioned work are usually identical to schedule variances in its related work. However, cost variances in the apportioned work are substantially different from the related work because of the dynamics of actual costs.	This measurement can employ any of the above six methods.
Level of Effort	Generally, level of effort tasks are those that support the overall project. Because these tasks are more time driven than task driven, whatever is set as the planned value is always the earned value, regardless of what work was done.	This method is generally not recommended to track earned value.

Table 1. *Different earned-value measures adapted from [3].*

earned value with the actual costs posted against the PMB yields the cost variance for the project.

Earned-Value Measurement Methods – With the PMB in place, performance measurements can be made. The specific methods to measure performance and earned value must be selected before the start of the project. A variety of methods to measure earned value have been proposed, and different methods are appropriate for different projects. Patricia W. Hurst presented the "binary reporting" method for earned-value measurement. According to Hurst, binary reporting is useful for projects of which their lowest level WBS work units are relatively small in effort, i.e., four to 80 staff-hours. Binary reporting maintains that work packages are in one of only two states: complete or incomplete. This gives the project manager a specific measure of the progress made with respect to the effort expended [9].

Other methods exist to measure earned value, including methods based on percent complete and weights applied to milestones [3]. Project and cost account managers have the responsibility to determine the most appropriate and effective method. Table 1 lists several categories of these measures.

Forecasting – Perhaps one of the most important benefits of earned value is its ability to forecast the final cost and schedule of a project. Successful forecasting is based on a foundation of a good baseline plan, tracking performance against that plan, and the commitment of upper management to use and act on the performance data. Several methods have been proposed to forecast project performance.

Q.W. Flemming and J.M. Koppelman present a forecast approach based on the work remaining, the cost performance indicator (CPI) and schedule performance indicator (SPI), and the actual costs for the project [3]. In this approach, the cost forecast is determined by computing the remaining work (usually the budget at completion minus the total earned value to date). This factor is then divided by either the CPI or the product of CPI and SPI.

This gives the remaining work with respect to the relative efficiency with which it will be completed. The actual costs expended to date are then added to this amount, yielding the forecasted cost of the project.

The schedule forecast can be determined graphically by examining the earned value and planned costs. Figure 3 displays a line graph in which budget is expressed on the vertical axis, and time is expressed on the horizontal axis. For a given status date, the BCWS, ACWP, and BCWP curves are plotted on the graph. Note that the date corresponds to the point at which the BCWP value intersects the BCWS curve. This date is compared with the status date, yielding the SV. This variance can be applied to critical path information to predict the potential completion date for the project.

Earned-Value Success Factors – As discussed earlier in summarizing ROI, it may be beneficial to consider a plan or road map to implement earned-value analysis in an organization (also see [11]). The following elements will contribute to such a plan.

- Ensure that the project is described by an appropriately detailed WBS with individual work packages at the lowest level.
- Create cost accounts for the project by ensuring that a specific organizational unit has responsibility for each work package.
- Establish a PMB, which incorporates schedule and budget information and against which progress will be measured.
- Identify the method that will be used to compute earned value.
- Identify reporting periods that are appropriate to the project, and identify the earned-value method selected.
- As each reporting period is achieved, measure values for BCWS, BCWP (the earned value), and ACWP. Use these values to compute other indicators, including the CV, SV, CPI, SPI, and other indicators of interest.
- Use the indicators to track and manage the schedule.

DoD and Industry Implementation Examples

ROI and earned value have been implemented with varying results in countless organizations over the past three decades. The following examples illustrate how ROI and earned value are being implemented in the real world.

In 1991, a software technology strategy for the DoD was drafted with three national objectives of note. The objectives included reducing lifecycle costs, reducing software problem rates, and increasing mission capability and interoperability. Over a period of five years, nearly 800,000 source lines of code were inspected. Using the number of major and minor defects identified and the total time to prepare for and inspect the documentation, an estimated ROI of a 4.48-to-1 ratio was computed [5].

Recently, the International Data Corporation conducted several in-depth economic analyses at major corporations. The corporations were inserting new technology to implement software process improvement. In computing ROI, the corporations emphasized four issues: rapid deployment on heterogeneous platforms, browser-based interfaces, ease of use, and leveraging openness and its impact on maintenance.

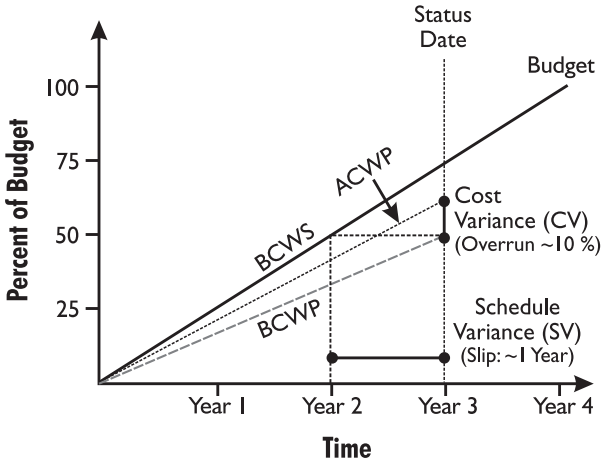


Figure 3. *Illustration of the earned-value curves BCWS, BCWP, and ACWP. Representative cost and schedule variances are shown with respect to a given status date. Adapted from [3].*

The projects were significant in size, ranging from $1.4 million to $4.2 million. A standard definition of ROI was used: ROI equals the amount above a dollar that was returned for every dollar spent in the implementation of the project. Two of the companies were Silicon Graphics and Amdahl. Over a three-year period, Silicon Graphics showed an ROI of 1,427 percent with a payback period of 0.18 years. Amdahl computed their ROI over three years to be 2,063 percent with a payback period of 0.13 years [10].

In 1993, the privately funded, multibillion-dollar IRIDIUM® satellite program began. Because the program was not federally funded, no government requirement was levied for the project to comply with the C/SCSC standard. However, the management group for the project implemented a tailored earned-value approach to manage the project. The earned-value approach was based on a product-type WBS. Earned value was embedded within the project's scheduling activities. Employees were rewarded based on the project's earned-value performance, which included cost and schedule performance and managing the critical path against key milestones. Therefore, this project created a unique approach to establish earned value as a valid and visible management tool.

Summary

ROI and earned value are relatively complex terms with enormous potential to enhance success at improvement activities or any other project. Both have been implemented, but neither consistently nor accurately, in the DoD and private industry. However, significant cost savings are available if their underlying principles are administered properly.

Earned value is related to ROI and is primarily based upon tangible estimates. It is an augmentation of traditional cost and funding management that provides the schedule management aspect that the traditional approach lacks. Earned value is a measure of "what you got for what you

paid" and is based on a foundation of work breakdown structures, cost accounts, performance measurement baselines, mathematical value computation, and schedule and cost forecasting. Any given method to compute earned value could be appropriate for some projects and inappropriate for others, so thought and planning are needed to select the best approach.

ROI is more difficult to uniformly measure and use in a practical manner than earned value. ROI can be computed by summing earned-value measures consistently and combining them with the less tangible estimates. Mathematically, ROI is the ratio of total savings achieved from an improvement effort to the total cost incurred to implement the effort. Some ROI estimates are easy to quantify, particularly those related to specific monetary expenditures and earnings; however, the definition of *costs* and *savings* could be expanded into multiple intangibles, which are much more difficult to estimate. Likewise, multiple methods to compute ROI may exist for each project and for each organization.

Ultimately, organizations that use earned value and ROI as a consistent strategy also will base their business tactics upon strong project management principles. The case for ROI measures can benefit from further study, including methods to compute ROI and identify intangibles and how to address them. ◆

About the Authors

Larry W. Smith is a software engineer at the STSC, where he has managed investigation and implementation of software reuse, software engineering, and other software process improvement activities for the Air Force and other DoD organizations. He has a bachelor's degree in electrical engineering from the University of Utah and a master's degree in computer science from Utah State University. He currently teaches project management and software engineering courses for the University of Phoenix.

Software Technology Support Center
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056
Voice: 801-777-9712 DSN 777-9712
Fax: 801-777-8069 DSN 777-8069
E-mail: smithl@software.hill.af.mil

A. Todd Steadman is a software development engineer at TRW Avionics Systems Division. He has provided software engineering technical services to the STSC for eight years, focusing on software project management technology, research, evaluation, and tool insertion in various DoD organizations and other federal organizations. He has written *CROSSTALK* articles and STSC technology reports on project management and cost estimation. He has a bachelor's degree in electrical engineering from the University of Utah and a master's degree in computer science from Utah State University.

TRW Avionics Systems Division
Ogden Avionics Engineering Center
1104 Country Hills Drive
Ogden, UT 84403
Voice: 801-625-8019
Fax: 801-625-8081
E-mail: todd.steadman@trw.com

References

1. Haley, T., et al., "Raytheon Electronic Systems Experience in Software Process Improvement," SEI/CMU-95-TR-017, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., November 1995.
2. Project Management Institute, *A Guide to the Project Management Body of Knowledge*, Project Management Institute, Upper Darby, Pa., 1996.
3. Flemming, Q.W. and J.M. Koppelman, *Earned Value Project Management*, Project Management Institute, Upper Darby, Pa., 1996.
4. Gilb, T. and D. Graham, *Software Inspection*, Addison-Wesley Longman, Ltd., Essex, England, 1993.
5. O'Neill, D., "National Software Quality Experiment – A Lesson in Measurement: 1992 – 1997," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, December 1998, http://www.stsc.hill.af.mil/CrossTalk/1998/dec/oneill.html
6. Desai, V., "When Computing ROI, Don't Forget the Intangibles," *Data Communications Magazine* (Web Edition), www.data.com, April 1998.
7. Flemming, Q.W., *Cost/Schedule Control Systems Criteria: The Management Guide to C/SCSC*, Probus Publishing Co., Chicago, Ill., 1992.
8. Department of Defense Instruction 7000.2, Performance Measurement for Selected Acquisitions, Washington, D.C., 1967.
9. Hurst, P.W., "Software Project Management: Threads of Control," *Software Engineering Project Management*, R. Thayer and E. Yourdon, eds., IEEE Computer Society Press, Los Alamitos, Calif., 1997, pp. 410-422.
10. Weiderman, N., et al., "Implications of Distributed Object Technology for Reengineering," SEI/CMU-97-TR-005, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., June 1997.
11. Flemming, Q.W. and J.M. Koppelman, "Earned Value Project Management: A Powerful Tool for Software Projects," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah, July 1998, p. 19.

Simulation: An Enabling Technology in Software Engineering

Alan M. Christie
Software Engineering Institute

This article suggests three reasons why the software engineering community could exploit simulation to a much greater advantage. First, the Office of the Secretary of Defense has indicated that simulation will play a significant role in the acquisition of defense-related systems to cut costs, improve reliability, and bring systems into operation more rapidly. Second, there are many areas where simulation can be applied to support software development and acquisition. Such areas include requirements specification, process improvement, architecture trade-off analysis, and product-line practices. Third, commercial simulation technology, capable of supporting software development needs, is now mature, easy to use, low cost, and readily available.

What Is So Great About Simulation?

Simulation can be applied in many critical areas. It allows issues to be addressed before they become problems. Simulation is more than just a technology because it forces one to think in global terms about system behavior and about systems being more than the sum of their components. Simulation can provide insight into the designs of processes, architectures, or product lines before significant time and cost have been invested and can be of great benefit in support of training. Simulation is being increasingly emphasized in the Department of Defense (DoD) community, where there is documented evidence that its impact on cost, quality, and schedule is nontrivial. I believe that the software engineering community needs to take a stronger role in exploiting the technology [1].

The DoD Emphasis on Simulation

The Office of the Secretary of Defense has recently initiated an effort focused on the use of modeling and simulation to support improvement of the acquisition process. Jacques Gansler, undersecretary of defense for acquisition and technology, states, "A directive which I issued this year [1998] requires the integration of modeling and simulation in our acquisition process—across functional disciplines—and throughout the lifecycle of systems. We are committed to reforming the acquisition system and recognize that an essential tool for accomplishing that reform will be modeling and simulation." [4] Although Gansler does not explicitly include the software acquisition process, there is no reason to doubt that software acquisition can benefit as much as any other DoD acquisition area, as will be explained further.

Gansler's remarks are reinforced by those of Patricia Sanders, director of defense, test, system engineering, and evaluation. In "Simulation-Based Acquisition," [5] she states that the DoD needs to become a smart buyer and that in evaluating what to buy, simulation will be a key component. She says that, "Without question, the Defense Department is moving toward greater use of simulation-based system acquisition." She indicates that,

> "The Defense Department envisions an acquisition process supported by the robust, collaborative use of simulation technology that is integrated across acquisition phases and programs. The objectives of Simulation-Based Acquisition (SBA) are to:
> 1. Reduce the time, resources, and risk associated with the acquisition process;
> 2. Increase the quality, military utility, and supportability of systems developed and fielded; and
> 3. Enable integrated product and process development from requirements definition and initial concept development through testing, manufacturing, and fielding." [5]

Sanders provides evidence from commercial and military programs to show that the use of simulation has had major positive impacts from the perspectives of cost, schedule, and productivity. Following are some of her examples.

> "Cost – … In the Joint Strike Fighter program, it is projected that virtual manufacturing techniques may save as much as 3 percent of the program's estimated lifecycle cost, which could be $5 billion.
> Schedule – The use of modeling and simulation tools and processes by the "big three" auto manufacturers has reduced the time from concept approval to production from 5 to 3 years. …
> Productivity – … It took 38 Sikorski draftsmen approximately six months to develop working drawings of the CH-53E Super Stallion's outside contours. In contrast, using modeling and simulation, one engineer was able to accomplish the same task for the

The Software Engineering Institute's work is supported by the Department of Defense.
Capability Maturity Model, CMM, and CERT are registered with the U.S. Patent and Trademark Office.

Commanche helicopter in just one month. …" [5]

Clearly, the use of simulation in the above examples is different from that in software development; however, there are sufficient parallels that would tend to indicate that similar advantages can be accrued in the software arena. For example, although physical mock-ups are not used in software development, early prototypes are used to the same advantage, e.g., determining system characteristics prior to large investments in implementation.

There are other common problems shared between the physical systems and software systems. Examples are

- The management of changing requirements and predicting the consequences of such changes.
- The development and optimization of effective processes through which the product is built.
- The estimation and tracking of project costs and schedules.

In addition, in 1998, the DoD developed an overall action plan to integrate the various simulation-based acquisition activities ongoing at the DoD. This action plan was developed by a joint SBA task force whose aim is "an acquisition process in which the DoD and industry are enabled by robust, collaborative use of simulation technology that is intended to integrate across acquisition phases and programs." [6]

The Need for Simulation in Software Engineering

Why can simulation enhance traditional software engineering? An important factor is that it provides insights into complex process behavior. Like many processes, software processes can contain multiple feedback loops such as those associated with correction of defects in design or code. Delays that result from these effects may range from minutes to years. The complexity that results from these effects and their interactions makes it almost impossible for human (mental) analysis to predict the consequences. Unfortunately, traditional process analysis does not shed much light on these behavioral issues, and the usual way to resolve them is to run the process and observe the consequences. This can be an extremely costly way to perform process improvement.

Assessing the Costs of Software Development

At an applied level, simulation can support project costing, planning, tracking, and prediction. In a competitive world, accurate prediction provides a significant advantage. If cost estimates are too high, bids are lost; if too low, organizations find themselves in debt. In this context, simulation can provide not only estimates of cost but also estimates of cost uncertainty. Simulation is a powerful tool to aid activity-based costing and can incrementally accumulate costs to an extremely fine degree of resolution. In addition, it can assess the uncertainty of costs based on the interacting uncertainties of independent variables [7, 8].

Supporting Metric Collection

Simulation is effective only if both the model and the data used to drive the model accurately reflect the real world. There is a tight connection between the model and the data in the sense that a simulation can only be executed if it is supplied with numerical drivers, which forces the developer to identify points in the model where these drivers are needed. For example, one set of data that needs to be entered in the model may be what percentage of design documents pass review and what percentage must be returned for further work. Thus, in the construction of the model, points where metric data must be collected fall out as a bonus. This approach forces the collection of metric data in a consistent sense from a systems perspective—it is not merely "nice to have" data. Often, too much or too little metric data is collected because the analyst does not have clear guidelines on what is essential.

Building Consensus and Communication

When changes to a process are proposed, experience is likely to be the most important influence. However, experience may not be enough to correctly assess behavioral changes resulting from process modifications. One person's experience may not correspond to another's, and subjective judgment comes into play as to whose opinion is correct. Usually, the person with greater authority wins. With the ability to quantify the effects through simulation, a much greater degree of insight and understanding can be brought to bear on the decision-making process. Therefore, simulation can be a significant influence in communication and consensus building. In this context, alternate process designs can be considered in a quantitative manner with respect to such issues as bottlenecking, resource availability, throughput, and costs. These analyses should result in processes that, once installed, will have a considerably higher probability of satisfactory operation.

A Discrete Simulation Model

There are many approaches to simulation. Some simulations are based on the need to visualize the airflow across a wing section, whereas others designed for combat or flight training need a virtual reality component. However, the types of simulations presented here use symbolic networks of linked elements that model processes or products. For example, it is possible to model entities flowing through the departments of an organization or model information flowing between a set of integrated software tools. Techniques such as discrete event simulation and systems dynamics are often used here.

To make concrete the type of simulations to which I refer, Figures 1 and 2 show components of a discrete simulation model ("discrete" because the entities that flow through the system are modeled discretely). The model was developed with the Extend tool [2] and depicts a call-center type of process for a computer security incident response team (CSIRT). CSIRTs, such as the CERT® Coordination Center at the Software Engineering Institute (SEI), support organizations that have been compromised by unauthorized computer intrusions or that wish to obtain information about guarding against such intrusions. CSIRTs have to communicate with many victimized organizations,

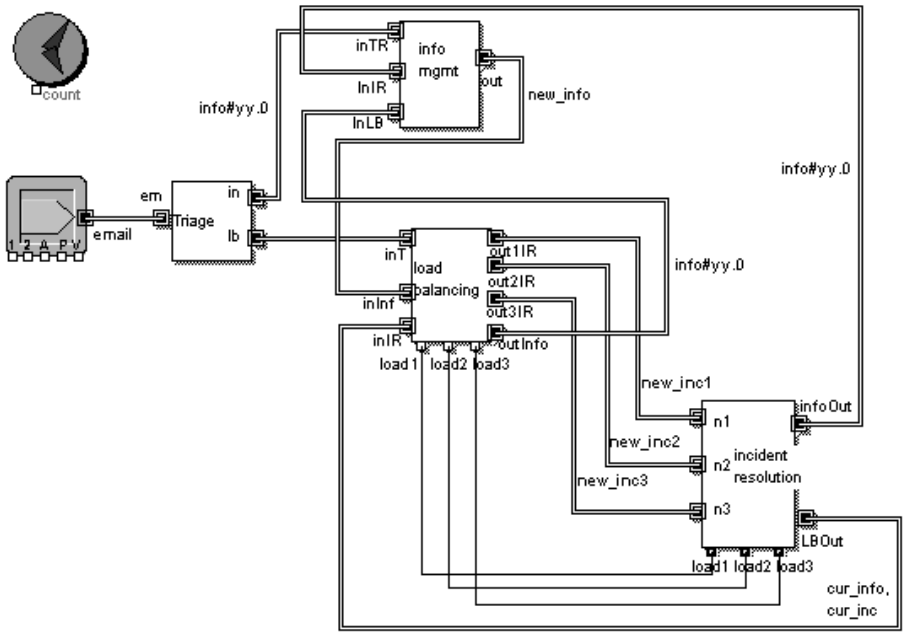


Figure 1. *Top-level view of a CSIRT process. This simulation models the flow of information through a CSIRT.* Triage *handles all incoming new messages.* Load balancing *distributes new requests for help among the incident-handling team.* Incident resolution *is the collective name for the team members who resolve incidents.* Information management *provides responses to routine requests for information.*

and one incident may be composed of numerous E-mail dialogs; hence, the need for a formal work-flow process to manage the large number of interactions while making sure that efficiency and responsiveness are maintained.

Figure 1 shows the top-level component of the incident-handling model.[1] New incidents are inserted into the process at the left, where they queue up to be handled by triage. In triage, E-mail is assigned to either the load-balancing function or to information management. In load balancing, the incidents are assigned to incident handlers based on the incident handlers' loads or their areas of expertise. Subprocesses take care of the details of the four activities identified in Figure 1 (triage, load balancing, incident resolution, and information management), which are all modeled in the simulation. Figure 2 illustrates the subprocess for the load-balancing area.

Upon running the model, various plots can be produced. In this example, both the queue in front of Incident Handler 3 (see Figure 3) and the number of completed E-mail completed for each of the incident handlers are plotted (see Figure 4).

Simulations such as this can be extremely useful for designing effective processes and for predicting the resources needed (both human and computer) so that the anticipated loads can be handled. This model contributed to generating synthetic incident data that supported performance tests on a CSIRT work-flow environment. Without such data, SEI would not have been able to assess performance at such an early state of the work-flow system's development.

For more background on technical issues associated with simulation modeling, consult [3].

Leveraging Simulation Across Applications

As illustrated in the next section, simulation can support a wide variety of applications; therefore, the marginal investment in simulation tools, training, and experience building diminishes as the technology is introduced to successively new applications.

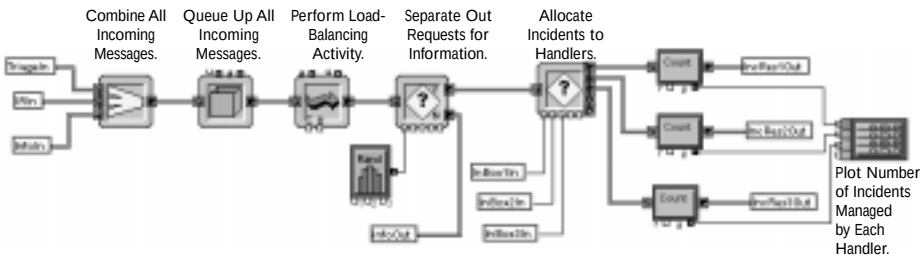
Target Applications for Simulation in Software Engineering

Simulation has been applied in many fields, such as aerospace and energy production, but to date, it has not seen broad practical application in software engineering. This may be because it is more difficult to accurately model human and organizational behavior than to model physical systems, or it may be that the emphasis on software process is a relatively recent phenomenon. Whatever the reason, it is unfortunate because the rewards from its use are myriad. In this section, I briefly review applications of simulation (in no particular order) and some of the benefits that can be obtained.

Requirements Management

Simulation can be extremely helpful in pinning down software system requirements early in the product lifecycle, particularly when examining temporal behavior. Simulation can mimic the performance characteristics of software components and their interactions, the effects of time delays and feedbacks, and of finite capacities and resource bottlenecks [9, 10]. The CSIRT example in Figure 1 illuminates these issues. Alternate architectures and designs can

Figure 2. *The load-balancing subprocess. The load-balancing activity attempts to assign incoming incidents to the appropriate incident handlers, i.e., those with lower current loads or those with expertise in the specific incident.*



be evaluated in a safe environment prior to implementation. In addition, requirements are rarely static but evolve as experience grows with product development. Thus, simulation is not only a valuable tool in defining the initial requirements but also can be used to test alternate modifications prior to their implementation. Finally, a system simulation can be viewed as a component of the requirements and can provide quantitative measures against which the target software system must comply.

The processes through which the requirements are managed also are critical. However, as far as modeling is concerned, such processes have much in common with other project management processes, e.g., design, development, and test. Thus, the discussion in the next section is relevant to requirements management.

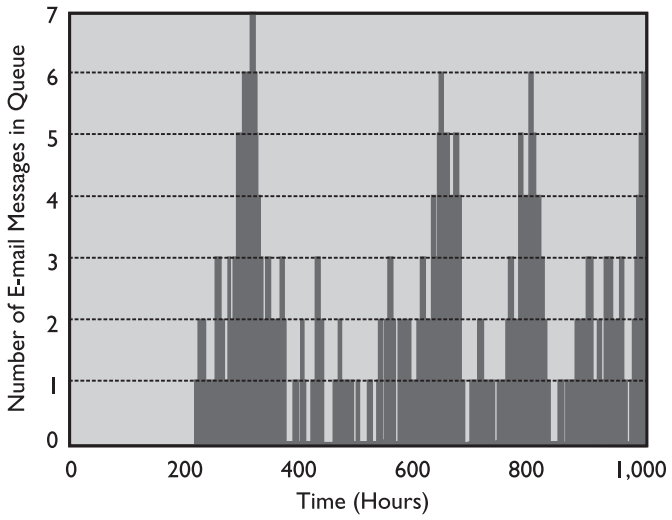
Project Management

Simulation can allow managers to make more accurate predictions about both the schedule and the accumulated costs associated with a project [11, 12]. This approach is inherently more accurate than costing models based on fits to historical data because it accounts for the dynamics of the specific process. With regard to schedule, simulation can account for dependencies between tasks, finite capacity resources, and delays resulting from probable rework loops. Some simulation tools also allow one to compute the accumulation of costs on an activity-dependent basis. These features are useful for generating proposals that are more accurate in cost and schedule and therefore more likely to keep a company in business.

Training

Because of the complex dependencies between attributes of organizational systems, these systems can respond in counterintuitive ways. (The classic example is Brooke's law, which states that hiring people late in a project can further delay the project.) Simulation can play an important role in sensitizing managers to the consequences of instabilities that result from the system feedbacks often inherent in badly designed organizational processes. Simulation-based training also can provide software development managers with the insights necessary to establish effective processes and to operate these processes in a stable manner. Thus, the focus is to train management in the design and operation of software processes, not in the technologies that support software development.

Figure 3. *The E-mail queue for Incident Handler 3.*



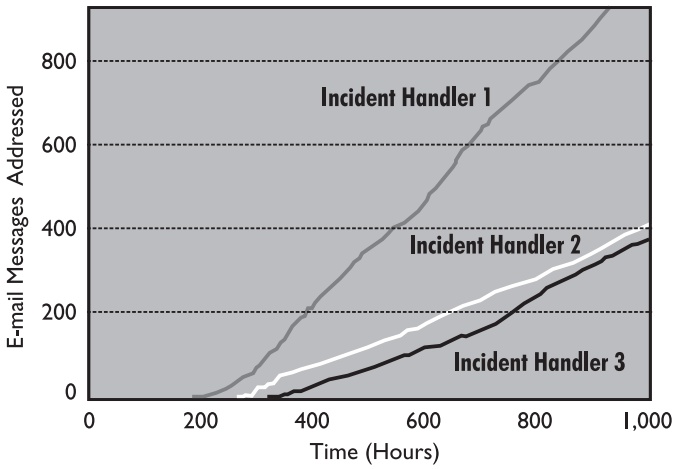


Figure 4. *The cumulative number of E-mail messages addressed by each of the incident handlers.*

Simulation-based training can be performed by individual managers who interact with the simulated software development activities. This person has control over certain control parameters (such as hiring rate and salaries), and the decisions made alter the course of the subsequent simulation history. Analysis of a training session can be performed after the session to see what went right and what went wrong in the decision-making process and to reinforce effective decision making.

To bring groups of managers to a central location for training can be both costly and time consuming. In the near future, such groups may be trained in a geographically distributed manner using a simulator with displays on the managers' local terminals. Interaction between trainees (which allows for joint decision making) can be provided through the use of collaboration technology. With the increasing interest in this technology, distributed training may soon become practical.

Process Improvement

Simulation can be used to support process improvement at all levels of the Capability Maturity Model® (CMM) but particularly at the higher levels [13,14,15]. Because simulation forces one to address metrics and process behavior in a methodical way (see "Supporting Metric Collection" section), one may argue that simulation can accelerate the introduction of process improvement. Consistent with the philosophy of the CMM, simulation capability at each CMM level incrementally builds on the simulation capabilities of the preceding levels and matches the needs of the software engineering practices at that level [16].

Traditionally, revised or new processes are improved through operational experience. This can be expensive and risky. Simulation can provide considerable insights into how a process will work prior to its implementation. These insights

can help the process designer assess alternatives and show that a specific process design performs in a manner that meets expectations. In this way, processes can be pretested, and buy-in is more likely obtained from management. Subjective criticisms are less likely, since quantitative simulation of validated models can produce specific and credible answers to perhaps hostile questions.

Architecture and Commercial-Off-the-Shelf Integration

Building complex software systems usually begins with addressing the system's architecture. Without a firm notion of how the major components of a software system interact, there is little likelihood that the system will reflect performance effectively. One would like to know early in the development cycle that such attributes as reliability, reusability, maintainability, portability, performance, and modifiability are above some acceptable level. There are complex dependencies between these attributes. For example, in improving performance, reusability might be sacrificed; or in improving portability, maintainability might require increased effort. Making trade-offs in this multidimensional space is not easy, but if they are not made at a high level of design abstraction, there is little chance they can be dealt with once coding begins. Simulation is a tool that can be used to examine some of these architectural trade-off issues [17]. Simulation can provide early insights into timing, resource usage, bottlenecking, and usability. In addition, one can rapidly gain insight into the implications of design changes by running simulations with varying independent parameters. Finally, one can assess sensitivities to parameter changes in a Monte Carlo (statistical) sense.

Product-Line Practices

Simulation makes considerable sense in the economic analysis of product lines. In particular,

> "Because product-line development involves changes in product composition and production, software size measures, such as lines of code, are not good predictors of productivity improvements. To estimate, track, and *compare* total costs of disparate assets, adaptation of other cost modeling techniques, particularly activity-based costing to asset-based software production, is needed." [18]

Some simulation tools incorporate activity-based costing such that, as entities flow through the simulated process, the cost associated with the processing of each entity at each stage can be accumulated. In this way, detailed cost predictions can be made with respect to different product-line strategies.

Risk Management

Projects are often vulnerable to risks resulting from things like requirements ratcheting, changing staff levels, funding cuts, and organizational disruptions. Simulation can help identify associated project risks early. By quantitatively predicting the consequences of alternate decisions, simulation can help design more objective, less risk-prone strategies. There also are risks associated with alternate system architectures or commercial-off-the-shelf integration strategies. By using simulation to examine the potentially complex interactions of alternate component configurations, the pros and cons of different design decisions can be identified.

Acquisition Management

Acquisition management is likely to be dependent on many of the practices described above, e.g., requirements management, project management, and risk management. Because all these practices can benefit from the use of simulation, acquisition management can, too. Specifically, simulation can help validate a contractor's estimates of costs and schedules and provide insight into the ability of the contractor's design to meet system requirements. Therefore, through the use of simulation, a project manager can predict potential contractor problems before they become reality. Simulation also has the effect of keeping the contractor honest in estimates of cost and schedule.

The subject of simulation-supported acquisition has been addressed in some detail by Walt Schacci and Barry Boehm [19, 20]. In their articles, they address the issues of how simulation can support the acquisition lifecycle. They give specific examples of potential applications and suggest that a research and development effort be established to explore issues such as virtual prototyping, incremental iterative acquisition supported by simulation, and the use of wide-area collaboratories.

Every Silver Lining Has a Cloud

As a cautionary note, it is well to remember that simulation is not a panacea. The predictive power of simulation is strongly dependent on how well the models are validated. Although many scientific and engineering fields can base their models on established physical law, organizational models have to deal with human and other less quantifiable issues. Not only is gathering data difficult when that data must come from human actors, the reproducibility of scenarios used to validate models cannot as easily be standardized as in experiments based on physical law.

Simulation is a simplification of the real world and is thus inherently an approximation. As indicated by S. Robertson,

> "It is not possible that a model is absolutely correct. Therefore, model [verification and validation] is concerned with creating enough confidence in a model for its results to be accepted. This is done by trying to prove that the model is incorrect. The more tests that are performed in which it cannot be proved that the model is incorrect, the more confidence in the model is increased." [21]

However, the usual alternative to simulation is to rely on human intuition, which Massimo Piattelli-Palmarini warns is often biased by "'mental blindspots' or 'mental tunnels' where we systematically make grave errors and get sidetracked into the wrong answer in certain kinds of problems." [22]

The State of Simulation Technology

Less than a decade ago, one could only develop a simulation by textual coding of the model. However, since the early 1990s, graphical simulation tools have become available. These tools

- Allow rapid model development by using, for example,
  - Drag-and-drop iconic building blocks.
  - Graphical element linking.
  - Syntactic constraints on how elements are linked.
- Are less error prone.
- Require significantly less training.
- Are easier to understand, reason about, and communicate to nontechnical staff.

Because of these features, network-based simulation tools allow one to develop large, detailed models rapidly. The focus thus becomes less on the construction of syntactically correct models and more on the models' semantic validity and the accuracy of their numerical drivers.

The simulation tools in today's marketplace are robust and reasonably inexpensive. Most tools cost in the range of $500 to $1,000. They therefore are accessible by organizations that wish to explore the applications described above. ◆

Acknowledgments

I thank Jim Withey, Bill Riddle, Anthony Earl, and Caroline Graettinger for their review of this article.

About the Author

Alan M. Christie is a senior member of the technical staff at the Software Engineering Institute. He actively promotes the application of process, collaboration, and simulation technologies to make software development more effective. He has extensive experience with process automation and barriers to its adoption. He published *Software Process Automation: The Technology and Its Adoption* (Springer-Verlag, 1995). He recently managed the implementation of a collaborative process support environment to meet the needs of computer security incident response teams. He also actively promotes simulation as an important element to understanding process behavior and to supporting process improvement. He has a master's degree in computer science and holds a doctorate in nuclear engineering.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Voice: 412-268-6324
Fax: 412-268-5758
E-mail: amc@sei.cmu.edu

References

1. Kirby, K. and R. Sawhney, "Simulation: Shifting the Competitive Edge for the Next Generation," *MDC Update*, University of Tennessee, Vol. 6, 1997.
2. http://www.imaginethatinc.com
3. http://www.pitt.edu/~wjyst/whatissim.html
4. http://www.acq.osd.mil/ousda/speech/modeling.html
5. http://www.acq.osd.mil/te/speeches/sanders/simbasedacq.htm
6. http://www.msosa.dmso.mil/sia-sba/sba_sia_documents.asp
7. Summary of CAPI-Developed Simulations for the U.S. Postal Service, http://idt.net/~capi99/usps.htm
8. Gardner, L.L., M.E. Grant, and L.J. Rolston, "Using Simulation to Benchmark Traditional vs. Activity-Based Costing in Product Mix Decisions," *WSC '94: Proceedings of the 1994 Conference on Winter Simulation*, pp. 1050-1057.
9. Belscher, R., "Evaluation of Real-Time Requirements by Simulation-Based Analysis," *First IEEE International Conference on Engineering of Complex Computer Systems*, IEEE Computer Society Press, Los Alamitos, Calif., November 1995.
10. Lerch, F., et al., "Using Simulation-Based Experiments for Software Requirements Engineering," N. Mead, ed., *Annals of Software Engineering*, Vol. 3, 1997.
11. Kellner, M.I., "Software Process Modeling Support for Management Planning and Control," *First International Conference on the Software Process*, Redondo Beach, Calif., 1991, pp. 8-28.
12. Abdel-Hamid, T. and S.E. Madnick, *Software Project Dynamics*, Prentice-Hall, Englewood Cliffs, N.J., 1991.
13. Raffo, D.M. and M.I. Kellner, "Using Quantitive Process Modeling to Forecast the Impact of Potential Process Improvements," *Proceedings of the 10th International Forum on COCOMO and Software Cost Modeling*, Pittsburgh, Pa., October 1995.
14. Hansen, G.A., "Simulating Software Development Processes," *IEEE Computer*, January 1996.
15. Tvedt, J.D. and J.S. Collofello, "Evaluating the Effectiveness of Process Improvements on Software Development Cycle Time via System Dynamics Modeling," *Proceedings of the 19th Annual International Computer Software and Applications Conference*, 1995.
16. Christie, A. M., "Simulation in Support of CMM-Based Process Improvement," *Journal of Systems and Software* (forthcoming).
17. http://www.sei.cmu.edu/publications/documents/97.reports/97tr029/97tr029chap03.htm
18. http://www.sei.cmu.edu/plp/modeling_costs.html
19. http://sunset.usc.edu/SAMSA
20. Schacci, Walt and Barry Boehm, "Virtual Systems Acquisition: Approach and Transitions," *Acquisition Review Quarterly*, Vol. 5, No. 2, spring 1998.
21. Robertson, S., "Simulation Model Verification and Validation: Increase the Users' Confidence," *Proceedings of the 1997 Winter Simulation Conference*, pp. 53-59.
22. Piattelli-Palmarini, Massimo, *Inevitable Illusions: How Mistakes of Reason Rule Our Minds*, John Wiley, 1994. Also http://public.logica.com/~stepneys/bib/nf/piattell.htm

Note

1. In these diagrams, double lines represent the flow of things, and the single lines represent the flow of numerical data.

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	***Tracy Stauder*** 801-775-5746 DSN 775-5746 managing_editor@stsc1.hill.af.mil
Senior Editor	***Sandi Gaskin*** 801-777-9722 DSN 777-9722 senior_editor@stsc1.hill.af.mil
Graphics and Design	***Kent Hepworth*** 801-775-5798 graphics@stsc1.hill.af.mil
Associate Editor	***Lorin J. May*** 801-775-5799 backtalk@stsc1.hill.af.mil
Editorial Assistant	***Bonnie May*** 801-777-8045 editorial_assistant@stsc1.hill.af.mil
Features Coordinator	***Denise Sagel*** 801-775-5555 features@stsc1.hill.af.mil
Customer Service	801-775-5555 custserv@software.hill.af.mil
Fax	801-777-8069 DSN 777-8069
STSC On-Line	http://www.stsc.hill.af.mil
***CROSSTALK* On-Line**	http://www.stsc.hill.af.mil/Crosstalk/crosstalk.html
ESIP On-Line	http://www.esip.hill.af.mil

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

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

E-mail: custserv@software.hill.af.mil
Voice: 801-775-5555
Fax: 801-777-8069 DSN 777-8069

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

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

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with *CROSSTALK*.

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

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

STSC On-Line Services: STSC On-Line Services can be reached on the Internet. World Wide Web access is at http://www.stsc.hill.af.mil. Call 801-777-7026 or DSN 777-7026 for assistance, or E-mail to schreifr@software.hill.af.mil.

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

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

Vinegar and Dye Pellets Not Included

I've learned that a fundamental rule of good writing and public speaking is to grab your audience with a witty, relevant opener that sets the tone for the remainder of the presentation. I intend to ignore this rule. Instead, I'll try to follow the standard established by software symposium speakers, which is to open with a corny joke that is only nominally related to the subject matter at hand:

Q: What is an Ethernet?

A: Something you use to catch the Ether bunny!

Ha-ha! And if that one has you doubled up on the floor in wheezing convulsions, hold on to that oxygen mask for another big jolt: That's the only joke in this column. And it's just as well, because what software developers do is no laughing matter, such as fixing the latest round of post-release flaws that were inserted while fixing the previous round of post-release flaws, or being tasked to implement your organization's objective of "leveraging team empowerment to facilitate synergistic initiatives," or attending weekly staff meetings.

However, the "Ether bunny" joke is somewhat related to this month's fatally grim software topic—software Easter eggs—because it affords me the opportunity to share some rambling, unrelated Easter nostalgia.

When I was a child, near this time every year, my siblings and I would lie awake and excitedly listen to the bunny in the next room romp through the house delivering hidden goodies. We knew that the next morning, we would be leaping from our beds the moment we heard those magical words: "There are rabbit doots behind the couch and the stereo wire's been gnawed through!" Dad would yell. "Who left the @!#% rabbit cage open last night?"

However, sometimes *another* bunny would come to our house and leave *chocolate* Easter eggs! But we won't be talking about the candy eggs that children snarf down on Easter Sunday before wiping their hands of the matter on the upholstery, forgetting the sugary feast by the following Thursday when it wears off and they finally go to sleep again. No, we'll be talking about the hidden bonuses found in software called *Easter eggs*, which is geek jargon for "the real reason a stupid word processing program takes up 20 MB of disk space."

But forget the cute-but-useless Easter eggs like cheat codes in "Doom," or the Magic Eight Ball in "Access," or 3-D developer credits in "Delphi." Instead, I'll reveal some simple key combinations that, when held down all at the same time, open some of the most powerful Easter eggs in programs you may already use.

Microsoft PowerPoint – (7+Esc+&+F11) Makes slides look less like they were created by an engineer. Limits number of boxes and arrows per slide to 25, and limits dissimilar pieces of clip art per slide to 12. Changes clashing color combinations like pink and orange to a more pleasing burgundy and neon purple.

Novell GroupWise – (Scroll Lock+Å+~+9) Interprets E-mail from management, as seen in the following verbatim example.

Original: "Seeing as our budgetary blah blah blah, we can foresee a shortfall in this, that, and the other, forcing us to mumbo jumbo gumbo. Your future input will prevent any unexpected yada yada yada."

Interpretation: "I'm too brilliant to have caused this mess, so the problem must be with you. Hey, now that I've hit that little envelope button, can someone show me how to run this E-mail through the postage meter?"

Lotus 1-2-3 – (Shift+4+#+✳) Puts a little frowny face [:-( ] next to numbers that are based on faulty logic. Figures that are massaged according to someone's political agenda are displayed in flashing neon green while the song "Liar, Liar, Pants on Fire" plays in the background (sound card required).

Norton AntiVirus – (Shift+§+ß+Æ) Cleans off not only malignant viruses but also *anything* politically damaging from your drive, including unsent hate mail, nonmission-related Web hits, and plagiarized material sources.

Microsoft Windows '95 – (Ctrl+Alt+Delete) Drastically improves system reliability and performance by zapping flawed, error-prone software. (For best results, insert a Linux installation disk in the "A" drive immediately prior.)

Adobe PageMaker – (Esc+Ctrl+F7+✉) Automatically generates a witty closing line when you can't think of one. – Lorin May

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