CrossTALK

The Journal of Defense Software Engineering

Discovering Lessons Learned

Sponsored by the Embedded Computer Resources Support Improvement Program (CRSIP)

January 2000
Volume 13 Number 1

Published by the Software Technology Support Center

Plot Your Course

by Reuel Alder
Software Technology Support Center

2020 A.D. The United States of America takes on another police action in Latin America and bombs the Chibcharian Embassy, killing several Chibcharian diplomats. Angered, the Chibcharians imprison thousands of Americans in retaliation. The President sends a carrier group and issues a demand to the Chibcharian Premier.

President: "Unless all U.S. prisoners are immediately released, we will begin to destroy your military installations."

Premier: ""Mr. President. I assume you are aware of the massive power outage that has just occurred in the Northeastern United States. All telephone service to the Southeast has been cut. The floodgates in the West have been opened. This is a demonstration of our capabilities. We are confident that many of your weapons systems will not function as a result of the hidden disabling code we have embedded in them. Our demands are simple. Recall your carrier group or we will signal your weapons to self-destruct. We demand a public apology and 5 billion dollars to compensate the family members affected. In addition, we demand the entire production of your Midwestern farmland in perpetuity."

The president reviews this threat with his technical advisors and discovers that the Department of Defense (DoD) has relied heavily upon commercial-off-the-shelf (COTS) components.

Early warnings were given. As Will Tracz writes in his article on architectural issues and other lessons learned in component-based software development (see page 4), it is important to know what COTS can do to you and beware of how you reconfigure your processes to meet COTS component capabilities. However, the real threat began when COTS development moved offshore because the United States only educated 17,000 software professionals per year, and the DoD's need for software far exceeded internal production capability. This exportation of software development had begun in the early 1990s, beginning with the country of Aidinia.

Aidinia was a model of off-shore development with production costs 1/10th of those in the United States. Aidinia applied disciplined software development concepts like the Capability Maturity Model (CMM) that the Software Engineering Institute developed. Its number of CMM Level 5 companies grew from five to 20 in 15 years before Chibcharia began capturing the market.

Following Aidinia 's example, Chibcharia educated more than 10,000 software professionals the first year of its program, and produced more software developers than the Aidinians by 2005. Five years later, all major corporations were doing business with Chibcharia, and nearly all DoD contractors purchased their COTS components from them. Contractors were rewarded for saving money, and COTS developed there at 1/20 the cost was the proven way to do it.

The security topics captured by Bryan C. Critenton in this issue (see page 27) are critical and will take a great deal of effort to resolve; however, they are only the beginning of DoD software challenges if core software development moves to foreign countries. This is a lesson we do not want to learn.

On the cover

acrylic painting by Alex Nabaum

Alex Naubam studied illustration at the Denver Art Students League and Utah State University. In addition to his work in graphic art, he works as an editorial illustrator in Salt Lake City.

ACQUISITION AND
TECHNOLOGY

THE UNDER SECRETARY OF DEFENSE

3010 DEFENSE PENTAGON
WASHINGTON, DC 20301-3010

26 Oct 1999

MEMORANDUM FOR COMPONENT ACQUISITION EXECUTIVES
DIRECTOR OF BALLISTIC MISSILE DEFENSE ORGANIZATION

SUBJECT: Software Evaluations for ACAT I Programs

It is DoD policy that software systems be designed and developed based upon software engineering principles. This includes the selection of contractors with the domain experience in developing comparable software systems, a successful past performance record, and a demonstrable mature software development capability and process. It also requires a software measurement process to plan and track the software program, and to assess and improve the development process and associated software product.

Software development and performance is an integral component of advanced defense systems. Accordingly, it will be a technical requirement for contract that each contractor performing software development or upgrade(s) for use in an ACAT I program will undergo an evaluation, using either the tools developed by the Software Engineering Institute (SEI), or those approved by the DoD Components and the DUSD(S&T).

At a minimum, full compliance with SEI Capability Maturity Model Level 3, or its equivalent level in an approved evaluation tool, is the Department's goal. However, if the prospective contractor does not meet full compliance, a risk mitigation plan and schedule must be prepared that will describe, in detail, actions that will be taken to remove deficiencies uncovered in the evaluation process and must be provided to the Program Manager for approval. The Deputy Under Secretary of Defense (Science & Technology) will define Level 3 equivalence for approved evaluation tools. The evaluation will be performed on the business unit proposed to perform the work. The reuse of existing evaluation results performed within a two-year period prior to the date of the government solicitation is encouraged.

This policy is effective immediately and will be incorporated into the current DoD 5000 series rewrite.

J.S. Gansler

Architectural Issues, other Lessons Learned in Component-Based Software Development

Will Tracz, Ph.D.
Lockheed Martin Federal Systems

Component-based software development, while reducing initial development time and effort, requires additional infrastructure and process support across the entire lifetime of an application. This paper summarizes lessons learned in dealing with commercial-off-the-shelf (COTS)-based architectures. In particular, it focuses on the technical and managerial issues associated with the acquisition, evaluation, selection, configuration, risk management, and evolution of software components.

"If the process hasn't changed, then the lesson wasn't learned."
—Ben Manning, Lockheed Martin Tactical Defense Systems

By using COTS components, software developers not only face increased opportunities to rapidly create new applications, but also face increased challenges to configure, integrate, and sustain these applications in the future. Arguably the technical key to success lies in the architecture that the system designers select for the components to be integrated into, as well as the middleware or "glue" that holds them together. Unfortunately, as proven by countless COTS-based project failures, project managers must take into consideration certain nontechnical factors, such as the volatility and flexibility of the requirements, the stability of the vendor, and the respective components that they supply. This paper describes some of the misconceptions that software developers and managers often have in planning for COTS-based systems [1]. First a series of myths are exposed in the light of reality of lessons learned from real world experience. This is followed by some rules of thumb for developing COTS-based systems.

This paper addresses COTS software issues, but the lessons learned equally apply to COTS hardware.

Operational Model

The COTS-based system acquisition and development model used by this paper assumes the following roles for the three major stakeholders:

1) The customer—who pays for the application to be built. Note that the customer also selects the developer/contractor/system integrator (e.g., through a competitive bid).
2) The developer—who builds the system out of COTS components. Note the developer may not always be the one to select the COTS components being integrated, as the customer may have stipulated them as part of the system requirements.
3) The COTS vendor—who supplies components to the developer and customer, along with its support and upgrades. It is important to point out that the customer needs to address the sustainment of the COTS-based system, or total ownership cost, because system maintenance can be adversely affected if certain architectural and procurement factors (e.g., licensing fees, security, and technology refresh) are not properly addressed early in the system life cycle upgrades.

The customer must address the sustainment of the COTS-based system or total cost of ownership (TCO) [2], because system maintenance can be adversely affected if certain architectural and procurement factors (e.g. licensing fees, security, and technology refresh) are not properly addressed early in the system life cycle.

Myths

This section contains a collection of COTS component lessons learned and is organized into two parts. The first subsection focuses on architectural issues. The second subsection focuses on management/procurement, or nontechnical issues.

Architectural Issues

The following myths deal with issues that the developer needs to consider when doing the initial trade-off analysis for a COTS-based system..

Myth 1: *It is important to know what COTS components can do for you.*
Reality: *It is important to know what COTS components can do to you.*

A system architect must always evaluate the build, buy, or modify tradeoff when determining how to meet the customer's requirements. COTS components, in general, provide certain functional capabilities at an extremely attractive *initial* cost. However, experience shows that functions come with limitations and implications. As the number of COTS components to be integrated increases, the dependencies and interplay among them becomes more complex, and can lead to intractable problems or difficult negotiations between different suppliers as to whose product is really at fault when things do go wrong. To complicate matters further, certain nonfunctional requirements, such as security, fault tolerance, or error handling, may not be uniformly supported by all components to the degree necessary to guarantee overall system performance. These potential hot spots typically form a list of risks that the architect must trade off in deciding the overall composition of the system under development.

Myth 2: *COTS-based systems can be designed top-down.*
Reality: *COTS-based systems are built bottom-up.*

COTS components facilitate a spiral development model in the sense that functionality can be quickly demonstrated in most

applications. In doing so, the customer benefits by early validation of requirements and the developer reduces risks by learning firsthand about the capabilities and configuration and integration difficulties associated with the components. Most architects understand this point and do not limit their choice of components by making design decisions too early. They recognize the need to remain flexible in trading off functionality across components until the components are fully proven and the integration mechanisms identified.

Myth 3: *An open-system architecture solves the COTS component interoperability problem.*
Reality: *There is no standard definition for "open system," and "plug and play" does not always work.*

Customers and developers clearly recognize the advantages of having plug-compatible components. They not only like having a choice of components, but knowing that if one component supplier goes out of business, there is another source for compatible components. It is debatable as to how successful open system initiatives have been (such as the Defense Information Infrastructure Common Operating Environment) [3]. Most will agree that when it works, it is great, but the number of plug-compatible components has yet to reach critical mass.

Myth 4: *You do not need to test COTS components.*
Reality: *You need to test COTS components more because you do not understand how they were built.*

It would be nice if all COTS components worked as advertised. But oftentimes there is a gap between what is advertised and what is delivered. Being that it is economically and oftentimes physically impossible for a COTS vendor to test all its products in combination with all other products under all operating conditions, subtle feature clashes can occur.

Furthermore, when developers are trying to leverage emerging technology, oftentimes marketing pressures force COTS vendors to deliver products with reduced capabilities along with the promise for increased functionality in future versions. Since the system integrator is usually responsible for the overall performance of the system, the system integrator should evaluate all components before they are selected for inclusion into the system.

Myth 5: *COTS products are selected based on extensive evaluation and analysis.*
Reality: *COTS products often are selected based on slick demos, web searches, or by reading trade journals.*

Because component-based architecture development is a relatively new field, systems integrators and customers still struggle with methods to keep abreast of technology advances and ways to determine which product best suits their needs. Oftentimes, in the rush to make a decision, the choice of COTS products is not made based on a strong business case or the total ownership cost. This problem has been around in various forms for a long time (e.g., GIGO [Garbage In, Garbage Out]) and ways of dealing with it (trade studies, test labs, independent product certification agencies) can be discriminators used by the customer in reducing risks associated with the acquisition of a COTS-based system.

Myth 6: *COTS components come with adequate documentation.*
Reality: *Features sell COTS components, not documentation.*

This myth may be thought of as a continuation of the previous two myths. The lack of documentation is a risk the system architect faces in determining the suitability of COTS components. In some instances the customer, upon being exposed to certain component features demonstrated in a certain (sometimes contrived) context, may place unnecessary or unrealistic constraints on the developer's implementation without adequate justification or flexibility in negotiating for different and possibly better components.

Myth 7: *You can configure a COTS-based system to meet your requirements.*
Reality: *You can configure your process to meet the COTS component capabilities*

The 80/20 rule applies to most COTS-based system efforts. The customer can satisfy 80 percent of its desired business process for 20 percent of the cost of a custom system (in 20 percent of the time). Most difficulties occur when a customer or developer thinks that the additional 20 percent is achievable at traditional software development costs. The cost of modifying COTS, or providing extra functions, is more difficult for the developer because it has little control or insight into how the COTS product was designed, documented, tested, or written/built. This information is usually proprietary and, in the light of upgrades and new versions, maintaining compatibility becomes a challenge. Most successful system integrators *never modify COTS*, and thoroughly understand the requirements (i.e., assuming an *all requirements are negotiable* approach). If the business case justifies modifying COTS components, then the developer should recommend that option.

Managerial Issues

The following myths deal with issues the customer considers when selecting a contractor to develop a COTS-based system.

Myth 8: *The processes COTS products support reflect industry best practices.*
Reality: *The process a COTS product supports often only reflects the market schedule and domain experience of the vendor.*

As much as COTS vendors would like the customers to believe that one size fits all, this is simply not the case (See Rule 12). Market considerations drive product offerings and most COTS component providers have product roll-out plans that include extended features and configuration parameters and hooks that allow tailoring and customization to support a better fit to best practices.

Myth 9: *You buy a COTS product.*
Reality: *You buy the right to use a version of a COTS product.*

COTS components provide immediate solutions at a fixed cost, but most applications have a life cycle that spans several releases of those components, which means that it is unrealistic (except in the case of hardware components) to expect the follow-on costs to be zero. In addition to the acquisition cost of

the components, the customer and developer need to explore the cost and level of support services as well as opportunities for commodity purchases.

Myth 10: *Vendors will fix problems in the current release of the product.*
Reality: *Vendors will fix problems in the next version of the product.*

As mentioned in the previous myth, the level of service one receives from the component supplier is negotiable. Unless the contract explicitly states it, the type of problem fixes one receives will be market driven (See Rule 6).

Myth 11: *If you are a large enough customer, you can influence COTS component suppliers.*
Reality: *The market influences COTS component suppliers.*

Again, the size of the current and future customer base drives the COTS component supplier in determining his response to user needs (See Rule 6).

Myth 12: *COTS-based systems are a panacea.*
Reality: *COTS components exacerbate inadequacies in the system development process by compressing the development schedule.*

To some, COTS components may seem like a silver bullet because they can provide faster, cheaper, and better solutions for:

- relatively simple applications.
- use in mature problem domains.
- using a small number of mature, unmodified components.
- proven integration mechanisms.

But not all applications fall into this category. The mere fact that applications are developed so quickly facilitates the possibility that the wrong application will be developed, the wrong COTS components initially selected, and the perceived short-term success will pave the way to long-term disaster.

Additional Myths

The following myths are relatively self-explanatory and reflect some of the points made in the rules of thumb stated in the next section, or previous myths.

Myth 13: *COTS components are free except for the purchase price.*
Reality: *COTS-based system sustainability issues overwhelm acquisition costs.*

Myth 14: *You can ignore vendor upgrades.*
Reality: *You lose support of back systems if you ignore vendor upgrades.*

Myth 15: *You can pay a vendor to modify COTS components to meet your requirements.*
Reality: *You can pay a subcontractor to modify COTS components to meet your requirements.*

Rules of Thumb

Rule 1: "*The cost of COTS is 1 percent of that of developed code.*"

This rule is attributable to Ed Feigenbaum of Stanford University and formerly the Air Force's Chief Scientist. To apply this rule, one would take the cost of a shrink-wrapped component and multiply it by 100 to get a rough approximation of the development cost for comparable function. Clearly there are other factors, such as the size of the customer base, for determining the cost of most COTS products, so one needs to use this rule judiciously.

Rule 2: "*The maximum shelf life of a COTS software component is two years.*"

This rule factors into determining the total ownership of an application in that all COTS components that have been configured and integrated together will probably have to be replaced two years after each was introduced into the marketplace. To complicate matters, each new version of a component might have additional dependencies and possibly introduce new, conflicting functionality. Also, the updates may not be released at the same time or validated with the same versions of other components, further complicating matters.

Rule 3: "*The half-life of COTS product expertise is six months.*"

This rule is attributable to Kurt Wallnau, Software Engineering Institute, who observed that with the fast-paced introduction of new product versions, as well as competing products, there is an unprecedented obsolescence associated with current technology. The inverse of this rule is that every six months you need to plan on evaluating a new version of a COTS product.

Rule 4: "*You need to evaluate COTS in an environment as close to the operational environment as is possible.*"

This lesson learned comes from too many bad experiences with COTS components that have been selected, designed around, and determined to have a pathological dependency that either completely precludes their incorporation, or makes the integration process much more complex and costly.

Rule 5: "*You can never make a 100 percent Diminishing Manufacturing Source-resident COTS-based solution.*"

Any commercial source of technology is outside the direct control of the customer. Consequently, for certain critical applications, system integrators and the customer must work together to take precautionary measures to ensure the sustainability of the application. These measures include paying a third party to store the design documentation and source code of the components or negotiating for the establishment of an open Application Program Interface in hopes of stimulating plug-compatible competition (i.e., a second source).

Additional Rules of Thumb

The following rules of thumb are self-explanatory. It is debatable which of the last two rules is more important, but it is clear that they play an important role in determining the success of any development effort.

Rule 6: *"The smaller the customer base, the higher the COTS cost and the better the service."*

Rule 7: *"The largest problem with COTS is its short life span."*

Rule 8: *"Stay away from the cutting-edge COTS products, unless it is the only way you can get the performance you need."*

Rule 9: *"By using COTS components you decrease development time and increase integration time."*

Rule 10: *"The selection of COTS components is a risk-mitigation or risk-creation situation."*

Rule 11: *"A COTS-based system may not be the cheapest solution."*

Rule 12: *"A COTS-based system will never completely or exactly satisfy a customer's need."*

Conclusion

Who is at fault for most COTS-based system failures? Is the customer to blame for expecting COTS to be a panacea? Are the developers to blame for not using good engineering judgement in identifying risks and opportunities to mitigate them? The customer must be flexible and must understand the short- and long-term tradeoffs with respect to certain COTS options. Current customer acquisition processes often force asking the wrong questions at the wrong time. In the case of government acquisitions in general, the customer neither has enough COTS-smart people nor has strong policy guidance.

But the customer should not take all the blame. Developers have been naive in trusting vendor vaporware claims and in underestimating the challenges of component configuration and interoperability. Fortunately, they are becoming savvier in their testing and integration capabilities. The ultimate solution, used by many of the leading system integrators, relies on setting up Integrated Product Teams consisting of the customer, the user, the developer, and the COTS suppliers. Such a forum often provides a venue where all stakeholders can better understand the requirements, their priorities, and the total ownership cost tradeoffs that are available with full insight on their short-term and long-term impact.

There are many COTS-based system development lessons learned. Unfortunately, the near-term trend seems to indicate that these lessons will be re-learned by many customers unless proper education, policy definition, and sharing of experience occur. Finally, it should be clear that a COTS-based system might not always be the best solution available. When all the factors are considered, a business case laid out, and a TCO study done, a custom implementation may be more cost-effective over the life of the project. ◆

About the Author

Dr. William Tracz is a senior software engineer for Lockheed Martin Federal Systems. Currently, he is lead architect for several COTS and Reuse Repository projects as well as the principle investigator on an internal independent research and development project focused on nonintrusive software integration mechanisms. Dr. Tracz also is an ad hoc member of the Air Force Scientific Advisory Board COTS panel, chairman of the Lockheed Martin Software Subcouncil Working Group on COTS Software and Reuse, as well as editor of ACM SIGSOFT Software Engineering Notes.

Lockheed Martin Federal Systems
Mail Drop 0210, 1801 State Route 17C
Owego, N.Y. 13827
Voice: 607-751-2169
Fax: 607-751-2169
E-mail: Will.Tracz@lmco.com
Internet: http://www.owego.com/~tracz

References

1. SEI, *COTS-Based Systems (CBS) Initiative*, available at http://www.sei.cmu.edu/cbs/index.html
2. Gartner Group, *Total Cost of Ownership: A New Tool for Controlling the Cost of IT*, http://www.info.edge.com/5509toc.htm
3. IPESO/DISA, *DII COE Defense Information Infrastructure Common Operating Environment*, http://dii-sw.ncr.disa.mil/coe/

New Guidebook Released on Systems Engineering Fundamentals

The Defense Systems Management College (DSMC) has released a new systems engineering guide, *Systems Engineering Fundamentals*, that it calls a more basic tutorial on systems engineering than DSMC's guidebooks released in 1982, 1986 and 1990. *Systems Engineering Fundamentals* was developed as a supplemental text to DMSC's systems engineering courses. DSMC uses the text in all its courses, and promotes it as a companion to the "How-to" Handbook published by the International Council of Systems Engineering (INCOSE).

The new guidebook is intended to be the systems engineering foundation for these courses; whereas, the INCOSE book is the application.

Additional description, as well as information on downloading a free pdf version or ordering hard copies, can be found at
http://www.dsmc.dsm.mil/pubs/gdbks/sys_eng_fund.htm

John Leonard of the Washington Military Area is the primary author.

Building a CM Database: Nine Years at Boeing

Susan Grosjean
Boeing Electronic Products

A Boeing organization developed an Oracle-based database to track problems during the life cycle of the Boeing 777 airplane. Over nine years, it has evolved from mainframe to web implementation as technology has become available. This article reviews some basic approaches to developing configuration management databases and includes resulting lessons learned.

The Need for a Database System

Boeing Electronic Products (EP) designs and develops some of the avionics[1] for Boeing Commercial Airplanes (BCAG).

About 1990, when the 777 airplane program was getting started[2] , EP was instructed to develop an electronic database to track and record problems encountered during design and development of EP electronics. BCAG wanted a database it could access. After the 777 was fielded, all airplanes were to use this database.

The existing problem reporting system was paper-based. One paper copy of each problem report (PR) was stamped "original." Multiple copies were made for each person assigned to work the problem. Each person would record his or her response by writing on the hard copy. Periodically, the Integrated Product Team (IPT) leader, responsible for a given piece of avionics, would call together all members of the team, known as the Engineering Review Board. These meetings could last for hours, trying to take all inputs and create one original PR. An easier way was needed to consolidate board members' input.

Requirements Definition and Implementation

Engineers, IPT leaders, and other potential customers did not seem to know what they wanted/needed in a problem reporting system. All the organizational requirements were surveyed (EP Configuration Management Plan, RTCA/DO-178B "Software Considerations in Airborne Systems and Equipment Certification," EP Software Quality Assurance procedures, etc.). Using the PR as a guide, the existing process was captured in a flowchart representation with a text description of each block. A requirements document was developed from this process document.

The numbering scheme for the PRs was used in conjunction with automation of the problem reporting process. A two-field scheme was used. The first field was an Avionics Product (LRU)[3] designation such as 7WEU (777 Airplane Warning Electronics Unit). The IPT leaders and auditors needed a list of all PRs for a desired component. A search on this first field would provide that list. The second field was a four-digit number that was assigned sequentially. Once a PR receives this two-field dash number, it cannot be deleted, so there is no break in the numbering sequence. This simplified the IPT leader's control and made any audit process easier.

Four text fields, as well as signature blocks, were associated with each PR. Each text field needed to accommodate extensive data (20,000 characters each). In the PR heading were smaller fields for pertinent data needed for tracking and reporting. The entry of valid data to these fields was enforced through the use of pull-down lists that provided all valid options for a given field.

Because of the size and quantity of PRs, the speed and capacity of a mainframe was needed. EP chose to use Oracle on a VAX mainframe rather than an IBM due to the availability of VAX programmers in EP. EP was unable to find an existing user interface to meet its requirements. User access to the mainframe was via dumb monochrome monitors or by loading an access icon to the personal computers.

A major implementation complication was the need to have five variations of the PR (i.e. variations on the fields and associated process). The struggle to agree on a uniform PR form and process was futile; each user organization wanted its own little corner of the world. The result was five different PR databases. Maintaining requirements, processes, procedures, updates, and training for five different databases was a nightmare.

Throughout the PR process (see Figure 1), e-mail messages were to be triggered to inform personnel and to assign tasks. There are now 43 possible messages for each PR. For instance, after an originator of a PR has completed the appropriate "title" and "description" fields, the system sends the PR as an e-mail message to those responsible for the product (component) in question. The length of these e-mail messages was a challenge, since they included the 20,000-character-length text fields. This will be solved by e-mailing only a brief message that includes a URL access to the web site that the assigned can access to see the complete PR and perform the applicable function.

The system also automatically changes the status of the PR as it moves through the phases shown in Figure 1. Portions of a PR are approved in each phase. After approval, the status changes and fields are locked, preventing changes by unauthorized persons. Engineers, configuration management specialists, and IPT leaders have different levels of authorization.

User Acceptance of the System

After implementation of the requirements and extensive testing, the database was given to the users. A program directive and a users manual were written. Hands-on training was also conducted—not an intensive course, but intended to acquaint the user with the database and convey some of the rationale used for its creation.

Function Keys

The users' first reaction was, "This database is not user friendly." No one liked using the function keys to move around the database. Many keyboards did not support the function keys we were using, and corresponding control keys, ^B or F10

= Exit had to be developed. These key strokes were displayed at the bottom of each screen.

Unlike the paper PR, the electronic version forced the entry of required information. The IPT leaders did not like the fact that the database forced everyone to do business the same way, and that it had checks and balances in place to ensure this practice.

But users liked the information the system provided: the position of a given PR in the life cycle, e-mail notification of PR status, who was assigned to work PR related tasks, and visibility into process bottlenecks. If a PR had languished on someone's desk for months, the IPT leader needed to know that. Sometimes the IPT leader was surprised to find that the languishing occurred on his or her own desk. Canned reports included listings of all PRs by responsible IPT leader or by assignee, and totals of the quantity of PRs by open and closed status.

Windows

Three years ago, Oracle offered Developer/2000, allowing a more user-friendly front end. We needed to retain the VAX, as by now all four text fields had increased to 65,000 characters each. With high-level management backing during the implementation of the front end, all Seattle-based groups adopted a single PR form. The groups also agreed with using basically the same process.

The Parts Engineering Organization in Seattle levied additional requirements, as it wanted a way to track and record problem reports involving obsolete parts.

Since a single part obsolescence is often common to several LRUs, a process enhancement was created to generate multiple corresponding PRs. A parent PR (the first PR written) is created and duplicate copies are made for each impacted LRU. These child copies retain a link to the parent. When the child PRs are closed, the parent PR can be closed.

Requirements expanded beyond Seattle to Irving, Texas—our manufacturing site. Irving has a problem-reporting system also implemented in Oracle. A requirement to transmit engineering problems between the Seattle and Irving databases was accomplished by developing an interface that transfers the PR from their database to Seattle's.

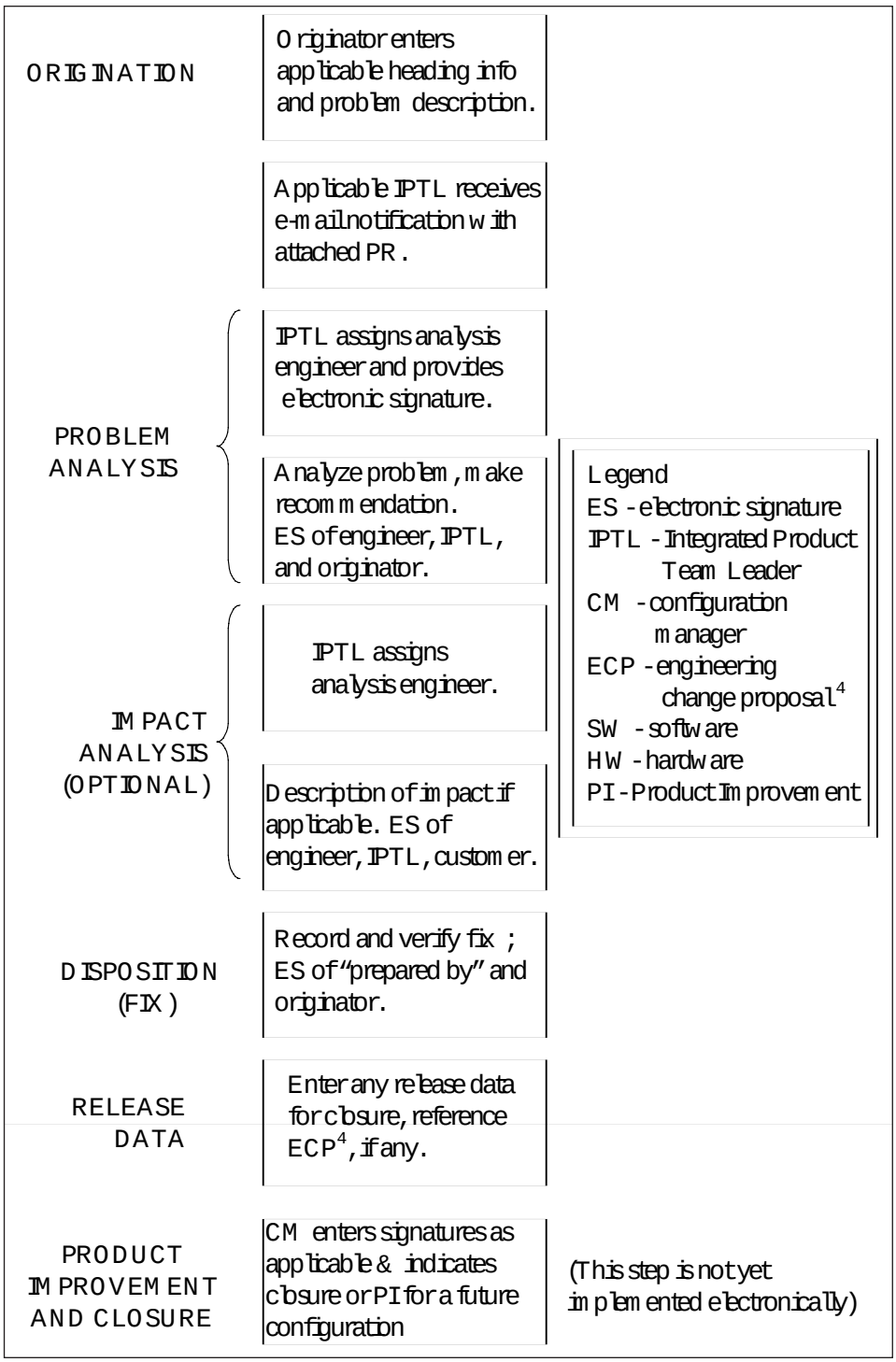


Figure 1. *Boeing Electronic Products (EP) Problem Reporting*

The Seattle system now has approximately 1,000 users with about 40,000 recorded problems. The database tracks and creates reports for any type of problem encountered by EP (product, test, obsolete parts, production) as well as lessons learned, action items, corrective actions, and PRs against the database itself.

Intranet

Access via the Boeing web is partially implemented. This provides access from more locations but with slower response times than those experienced by users who access the system directly using the software application windows front end. The database is migrated off the VAX to a Unix operating system.

Lessons Learned

You may consider commercial off-the-shelf tools that may meet most of your requirements or investigate other organizations or divisions using a tool similar to your requirements. Implementing this system has been a lot of work spread sporadically over nine years. Remember, EP had in-house Oracle developers; many organizations do not have similar resources.

Establish a team to consider new requirements. One result of nine years of experience is EP's use of an Engineering Review Board that helps determine if a change or enhancement will benefit all users. This database has a PR type, which enables the users to write a PR against the PR system. These PRs include problems encountered and enhancements.

Designate a focal point to write and transmit requirements to the database programmer. This should be one or two people who are familiar with the database and who interface well with the programmer. Once approved by the Engineering Review Board, the focal point writes or rewrites the requirements to ensure they are easily understood by the programmer. The change/enhancement is added to the users manual and if the change/enhancement is substantial, training is scheduled.

Consider the importance of your testing approach. For the first four years, a separate test database was maintained. Later, the test database was dropped because of the work required to maintain it. Testing is now performed on the production database. This has been a cost-effective compromise that has resulted in few bugs.

Make sure there is room to grow. In this case, text fields have grown from 20,000 characters to 65,000. Other organizations may decide to use the database once it has proven itself. Also, the more you give users, the more they seem to want.

Sponsorship can make it easier. Once senior management saw the benefit of using a standard process, a single PR, and a single database, EP no longer had to maintain separate requirements, processes, procedures, and updates.

Technology improvements can make it easier. By providing pull-down and pop-up lists within the PR fields, diverse users could live with the generic PR.

Training is essential to the effectiveness of the database and the user. It saves users hours of time by not having to struggle with an unknown element, and gives them hands-on experience.

Help lines and training manuals are important in helping users further understand what they are expected to do. Help lines appear throughout most of the screens to inform and ensure that the PR is completed accurately. The training manual for this database is set up for step-by-step instructions with visual aids.

Summary

Boeing's need for a CM database has been outlined. Defining the database requirements in specific terms for design and implementation presented some challenges. The implementation presented user interface challenges. The system has evolved to meet new requirements and to exploit new technology.◆

About the Author

Susan C. Grosjean is a configuration management specialist with the Electronic Products Group. She has 20 years of configuration management experience at Boeing. Prior to developing the database described in this article, she developed status accounting databases for the B1-B and B2 bombers.

Boeing Commercial Airplane Group
P.O. Box 3707, No. MS OU-47
Seattle, Wash. 98124-2207
425-266-3731
Susan.Grosjean@PSS.Boeing.com

Notes

1. Electrical and electronic devices in aviation, including the software embedded in those devices.
2. For BCAG, the 777 airplane represented an unprecedented software challenge in terms of the size and complexity of the airplane's systems. The 2.5 million lines of newly developed software were approximately six times more than any previous Boeing commercial airplane development program. Including commercial-off-the-shelf and optional software, the total size is more than 4 million lines of code. This software was implemented in 79 different systems produced by various suppliers throughout the world. *CrossTalk* January 1996, "Software Development for the Boeing 777."
3. Line replaceable unit is the terminology for the circuit board, drawer, or cabinet that can be removed and replaced when hardware failure occurs.
4. Class I Changes are called Engineering Change Proposals (ECPs) and consist of changes that are so extensive that additional customer funds must be contracted to implement the change. Changes smaller that Class I are called Class II.

Talk to CrossTalk

We welcome reader comments regarding *CrossTalk* articles or matters pertaining to software engineering. Please send your comments and Letters to the Editor to crosstalk.staff@hill.af.mil or mail to

OO-ALC/TISE
Attn: *CrossTalk* staff
7278 Fourth Street
Hill AFB, Utah 84056-5205

Please limit letters to less than 250 words. Include your name, phone number, and e-mail address with any letter. We will withhold your name upon request.

The CM Database: To Buy or to Build?

Reed Sorensen
Computerized Thermal Imaging Inc.

For some organizations, it makes sense to develop a database system that supports reporting, auditing, problem tracking, and certain project management (PM) functions rather than buy a commercial configuration management (CM) product that additionally provides capabilities such as version control, build, merge, and conflict identification. This article provides examples of organizations that have developed a CM database and discusses the relationship between the PM and CM disciplines.

Project managers and program managers need ready information about the status of the system they are maintaining/developing to answer the questions:

- What are the pieces of the system?
- How do the pieces relate?
- Which pieces are being purchased?
- Which pieces are under maintenance contract?
- What am I spending to maintain a component?
- How many change requests are outstanding and against which components?
- What is the status of a change?
- What components present the greatest business, schedule, and technical risks?

These are a sample of many questions that can be asked.

Project Management and Configuration Management

A search for answers may bring a person to more than one management discipline, namely the project management discipline and the configuration management discipline. EIA Standard IS-649 [1][1] notes the relationship between these disciplines, and Ovum observes that the relationship will be reflected in the tools that support PM and CM [2].[2] While tools can be bought to help answer the questions, the challenge is in finding one that most closely meets your functional, budgetary, and cultural requirements.

This article assumes you have determined that most of your questions are of a CM status reporting nature. Tables 1 and 2 list the kind of information typically attributed to PM and CM. The tables also enhance understanding of the relationship between the PM and CM disciplines if you modify the PM information table by substituting the words "problem/change" for "project." Often a problem/change becomes a project if it is big enough.

actual progress with respect to planned progress
individual resource profiles (skills, labor rate)
work breakdown structure codes and associated fields
dependencies for project tasks
project calendar
schedule impacts due to increased workload
flexible cost reporting (e.g. by account, department)
critical path
Gantt chart view of project schedule

Table 1. *Representative PM Information*

list of configured items
map changes/problems to engineer assigned
status of problem/changes (open, deferred, closed)
report progress on problem/changes
version information
differences between versions
file revision history

Table 2. *Representative CM Information*

Buy vs. Build

A manager with access to in-house software development expertise may, at some point, consider the expense and effort of evaluating, procuring, customizing, and maintaining a commercial-off-the-shelf product (COTS) vs. the expense and effort of building and maintaining a tool. While COTS tools usually include some kind of database, they are not specifically a database. For the manager who is focused specifically on the kind of functionality that a database system, including a graphical user interface and database engine, can provide, the COTS CM tools are not necessarily the most attractive option. Often an in-house developed database is a better match functionally. Some may be more easily procured, if the in-house development resources are available.

Aside from functionality and economics, cultural issues may sway a manager to build rather than buy the tool. The argument, "We are a development organization; why pay outsiders to do what we can do?" carries some weight. Your people may simply want to build their own tool vs. learn a COTS product, so there is a morale issue or bias.

The vendor of a COTS CM product might point out that the vendor's product can be customized to any process, that it will be more mature than a home-grown tool (better documentation, ready training, fewer defects) and that maintenance costs will be lower. There is the potential additional advantage of having all the data and CM functionality in a single system as noted by Ovum's Clive Burrows:

> Gathering management information is greatly simplified if change features are part of the CM system—without them, complex cross-references between different databases are required, and full navigation and searching may not be possible.
>
> Unfortunately, many CM vendors have developed their own add-on capability in this area using new development tools, different databases, and even a different style of user interface. In some cases, the only area of commonality is the product "badge" created by the marketing department [3].

In practice, the buy vs. build decision is not an either/or solution. This author sees a lot of organizations that use COTS products for some CM requirements and "roll their own" CM database to handle status accounting and project management requirements. Here are three such organizations.

Three Air Force Examples

1. In the 1980s, the Automatic Test Equipment projects in TIS at Hill AFB had developed an in-house database in dBase IV, a nonrelational database tool available from Inprise Corporation (formerly Borland). In 1997, TIS used its own software developers to migrate the database to a Microsoft Access implementation to control the status accounting requirements for configuration management.

   When TIS was contracted to

configuration manage the GTACS [3] software, it followed the Program Office mandate to use and implement CCC Harvest. The GTACS software development community required a robust code management tool. After a year of working with Harvest, TIS chose to keep its Microsoft Access database for another year and then migrate it to Oracle and also retain CCC Harvest. The bottom line was that it needed Harvest for the code management and it liked its in-house developed database for information management.

2. CIDSS at Peterson AFB maintains the software for the Space Environmental Support System. It chose Microsoft Access to implement Project Logging and Tracking Tool (PLATT), a database that tracks requirements submitted by the customer as well as internally generated software maintenance tasks. While not experienced with Microsoft Access as a development tool, one CIDSS person with strong software development background and the ability to learn implemented the database. Version control of software code at CIDSS is implemented using Configuration Management System by Digital Electronics Corp.

3. TISHB at Hill AFB develops operational flight software for the F-16 multirole fighter. For the past seven years it has used a database that TISHB developed to track changes to the software and associated test stands. The approach has been successful for at least two reasons:
1) the Sybase Server used to implement the database is used for several other applications and was already available when the buy or build decision was made.
2) the responsiveness of in-house Sybase expertise means changing requirements are implemented readily. As a capability becomes obsolete, it is deleted from the system. Version control, file merging, and build management are implemented at TISHB using Concurrent Versions System freeware with a user interface that TISHB developed.

There is a common thread in these three scenarios. All of the above organizations are using some COTS product for CM, but also chose to implement a database on their own.

Recommendations

Define your process and requirements.
Whether you buy or build, half the battle is defining your requirements [4]. Requirements drive your database design and implementation, or your evaluation and customization of a COTS product. Documenting your current approach to CM is part of the definition task. You may also have in mind a desired future process that differs from the current process; if so, also document the future process.

Walk, do not run.
As a manager, can I live with my current mode of operation, and if so, for how long? Boeing's approach to developing a database over several years was successful for them [5]. You will need to trade off the time required to implement a database vs. the cost and quality issues to implement that database. Faster implementation means more initial expense (purchase or development) or less quality (less functionality and/or more bugs).

Use as little automation as practical.
Some level of automation will be appropriate in managing the data that answers the questions posed in this article's introductory paragraph, but you will want to use as little as practical. This will maximize your answers and minimize the cost of maintaining the automation. Do not maintain a high-end tool when a spreadsheet will provide the answers.

Conclusion

As a manager, do I have internal expertise that is available for implementing a solution based on my well-defined requirements? Do I have the needed database development tools available? If so, the build option is attractive. Table 3 is a representative list of database development tools.◆

About the Author

Reed Sorensen serves in configuration management, quality assurance, and technical publication roles at CTI (Computerized Thermal Imaging Inc.), which deploys thermal imaging and associated technologies for use in medical screening, diagnosis, and patient management. Sorensen has more than 20 years experience:

1) developing and maintaining software and documentation, and
2) improving configuration management and documentation processes.

He has published articles in *CROSSTALK* on various software related subjects.

Computerized Thermal Imaging Inc.
476 Heritage Park Blvd., Suite 210
Layton, Utah 84041
Voice: 801-776-4700
Fax: 801-776-6440
E-mail: reeds@siinet
Internet: http://www.cti-net.com

References

1. *EIA/IS 649 National Consensus Standard for Configuration Management*, August 1998, page 1
2. *Ovum Evaluates: Configuration Management*, 1999, Page 12.
3. Burrows, Clive. "Configuration Management: Coming of Age in the Year 2000." *CrossTalk*, March 1999.
4. See Susan Grosjean's *Building a Configuration Management Database: Nine Years at Boeing*, in this issue.
5. IBID.

Product	Vendor	http://
DB2	IBM Corporation	www.softwareibm.com/data/db2
Informix	Informix Software Inc.	www.informix.com
dBase	Inprise Corporation	www.inprise.com
Access	Microsoft Corporation	www.microsoft.com/office/access
SQL Server	Microsoft Corporation	www.microsoft.com/sql/?RLD=183
VisualFoxPro	Microsoft Corporation	msdn.microsoft.com/vfoxpro/
Oracle	Oracle Corporation	www.oracle.com
Sybase	Sybase, Inc.	www.sybase.com

Table 3. *Some database development tools and vendors that may or may not be useful for your organization* [4]

Notes

1. Configuration management principles underlie sound business practices used throughout industry and government to provide: . . . Access to accurate information essential to the product's development, fabrication, production, use, maintenance, procurement, and eventual disposal.
2. ...in future the scope of what is considered to be CM will undoubtedly include strong links with project management systems.
3. *Ground-based C2 elements of the Theater Air Control System* supporting air operations performed by the Combat Air Forces.
4. This is a representative list, not an exhaustive one. The list is for information only; no endorsement of these products or vendors is implied.

Call for Articles

If your experience or research has produced information that could be useful to others, CROSSTALK will get the word out. We welcome articles on all software-related topics, but are especially interested in several high-interest areas. Drawing from reader survey data, we will highlight your most requested article topics as themes for future issues. In future issues of CROSSTALK, we will place a special, yet nonexclusive, focus on:

CMMI
June 2000
Submission deadline: Feb. 1, 2000

Personal Software Process and Team Software Process
July 2000
Submission deadline: March 1, 2000

Object-Oriented Technology
August 2000
Submission deadline: April 3, 2000

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

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

Ogden ALC/TISE
ATTN: Heather Winward
7278 Fourth Street
Hill AFB, Utah 84056-5205

You may e-mail articles to features@stsc1.hill.af.mil.
or call 801-775-5555 DSN 775-5555.

Learning: The Engine for Technology Change Management

Linda Levine
Software Engineering Institute

This is the second part of a two-part article. Part one, published in November, explored the adaptations needed in the process movement and knowledge-creation approaches to achieve the vision of a learning organization. Part two looks at learning in practice by examining some frameworks and tools that pull together process, knowledge management, and technology to support learning and effective change.

This author considers the role learning typically plays in technology adoption through lessons learned and training, extending more recently to the use of repositories. This is contrasted with models that support learning and successful change more actively—through improvement frameworks and tools—as illustrated by the examples of IDEAL℠, an established model for software process improvement,[1] and the IDEAL℠-based New Technology Rollout (INTRo), a web-based process guide for technology change management.[2]

Learning is the fuel for technology change management. In turn, the creative exploration and exploitation of technology, knowledge, and processes are critical ingredients for realizing learning organizations. Once we recognize these interdependencies and the need for knowledge integration, we will be able to move away from education and training-by-firehose instruction toward the next generation of approaches and mechanisms to support communication, coordination, and collaboration.

Why is Learning Important?

Technology change management is not an isolated activity but a process that touches many of the socio-technical activities at work in an organization. This bigger picture of technology change management includes business and work processes, and technical systems, as well as processes related to group dynamics and collaboration.

The connection is clear. When we ask people to change how they do their work, as we do in improvement or technology adoption efforts, we are asking them to learn. If you pay attention to how people learn, you will be capable of more effective change management. Learning and technology change management reinforce one another. If you are smart about how you manage change, you will help make your workplace a learning organization, and that will pay off in many ways.

Learning is the keystone for dealing with the high number of failed change efforts, the rapid rate of change in information technology, and the need for new organizational and management constructs. Why is learning important?

- Approximately 70 percent of business process re-engineering efforts, or redesigns, fail [1,2]. The number and degree of failures would decline if we paid attention to connections between technology adoption and learning. Resistance is typically trivialized and resistors are seen as people that need handling. In fact, resistance can anticipate and surface flaws in intent, design, and implementation, and be a predictor of problematic and high-risk endeavors [3].
- Work groups of the 21st century will manage change in dynamic situations. Older freezing and refreezing metaphors from organizational development are inadequate [4]. Multimedia technologies and practices supporting process change, modeling, simulation, and collaborative and distributed work are key. Skill sets in the new work force that allow for flexibility, speed, and experimentation will be prized. We must learn how to learn.
- Traditional management constructs are outdated. Managers and practitioners express different kinds of fearfulness about adopting certain innovations, specifically process automation, workflow technologies, and groupware [5,6]. We must address the incompatibilities between collaborative technologies and organizational hierarchies and bureaucracies if we are to ease the transition to newer forms of work groups, including high-performance teams, self-directed, autonomous teams, and integrated product (process or practice) teams.

What About TCM in the CMM?

We can already see a rethinking of technology change management in proposed changes to the Capability Maturity Model for Software (SW-CMM v1.1.) Draft C of the SW-CMM, enlarges the scope for technology change management: the purpose of Organization Process & Technology Innovation "is to identify process and technology improvements and innovations that would measurably improve the organization's software processes, thereby helping achieve the organization's software process improvement goals."

The transformation of Technology Change Management into Organization Process and Technology Innovation is a step toward a more robust and innovative approach. However, much remains to be understood about these activities, especially about how they are operationalized.

Are we ready for technology change management and learning organizations? Yes, more than ever. We are at a watershed with the potential to get leverage from experience in process improvement, our intellectual investment in organizational learning, matching our interests with enabling information technology. But first, how are we learning now?

Learning: The Current State

If we were to describe how learning takes place in our organizations, chances are we would say it occurs through:

- training

The Capability Maturity Model for Software and SW-CMM are registered in the U.S. Patent and Trademark Office.

- professional development
- lessons learned documents
- war stories
- process asset libraries (PALs) or repositories

Training and professional development are usually associated with individual learning. Project learning can be captured when lessons are documented; war stories are sometimes shared informally with others. Such stories convey valuable, undocumented learning (that may be imprudent to record) gained during the life of a project. When and where are these lessons applied? A good question, and one most organizations fail to address. Process asset libraries represent our best current attempt at information sharing and organizational learning, where artifacts (process and method descriptions, plans, standards, policies, templates, forms, etc.) are made available for adaptation and reuse through repositories. Too many of these libraries, however, degrade to online file cabinets or archives. The repository's role as coordinating mechanism—a forum and space for information sharing and exchange—in the larger learning environment is never realized.

Our understanding of the organization as a structure, as building and edifice, drives how we tackle learning and work. We build complementary forms in the same image as the organization—whether these forms are products or projects or teams—in order to handle problems. The challenge awaits to envision and internalize a newer understanding of the structure of the organization as a permeable, flexible, virtually networked locus of activity. If we accept this challenge and the shift it represents, gradually we will gravitate to solutions that emphasize transaction and interaction, and serve as vehicles for communication and collaboration. As our experience base grows, so will our ability to work in solution spaces where learning is valued.

Active and Interactive Learning: The Desired State

> "The lecture is the most inefficient method of diffusing culture. It became obsolete with the invention of printing. It survives only in our universities and their lay imitators, and a few other backward institutions…. Why don't you just hand print lectures to your students? Yes I know. Because they won't read them. A fine institution it is that must solve that problem with platform chicanery" [7].

The models and mechanisms for organizations of the 21st century will support learning by doing—communication, coordination, and collaboration—in an interactive mode. Now, we primarily learn from discrete events in which we are involved, and we have scant understanding of the work and learning of others. Consequently, we fail at lessons applied on a triple score.

1. Few incentives explicitly encourage lessons applied and information sharing;
2. Even fewer mechanisms transfer knowledge across teams and parts of an organization;
3. We rarely reflect on the learning process. Chris Argyris offers a valuable characterization.

Argyris distinguishes between single-loop learning, which asks a "one-dimensional question to elicit a one-dimensional answer." He offers the example of how a thermostat measures temperature against a standard setting and turns the heat on or off accordingly. He contrasts this with double-loop learning, which "takes an additional step or, more often than not, several additional steps. It turns the question back on the questioner. . . . In the case of the thermostat, for instance, double-loop learning would wonder whether the current setting were actually the most effective temperature at which to keep the room and, if so, whether the present heat source were the most effective means of achieving it. A double-loop process might also ask why the current setting was chosen in the first place. In other words, double-loop learning asks questions not only about objective facts but also about the reasons and motives behind those facts" [8].[3]

Double-loop learning describes the active learning we have been emphasizing. The benefits for knowledge integration and transfer across organizations and communities are also evident; this was the topic for Part One of this article. The practical question remains: What early examples do we have of frameworks or tools that try to pull together process, knowledge management, and technology to support organizational learning? Two such examples are IDEAL (Initialing, Diagnosing, Establishing, Acting, Leveraging), a high-level model for software process improvement, and the IDEAL-Based New Technology Rollout (INTRo), a web-based process guide for technology change management. INTRo was designed with the principles of organizational learning and knowledge management in mind. We consider these examples and the role that learning plays in each.

The IDEAL Model

The IDEAL model was conceived as a life cycle model for software process improvement based upon the CMM for software. The model provides a disciplined engineering approach for improvement, focuses on managing the improvement program, and establishes the foundation for a long-term improvement strategy [9]. Following the phases, activities, and principles of the IDEAL model has proven beneficial in many process improvement efforts. The model consists of five phases listed below.

Development of IDEAL was prompted by requests from people who were engaged in Software Process Improvement (SPI) initiatives based upon the SW-CMM. The original model incorporated results of fieldwork the Software Engineering Institute (SEI) had conducted with early SPI adopters. Designed to satisfy specific needs, its focus was initially

Table 1: *The Five Phases of IDEAL*

I Initiating	Laying the groundwork for a successful improvement effort
D Diagnosing	Determining where you are relative to where you want to be
E Establishing	Planning the specifics of how you will reach your destination
A Acting	Doing the work according to the plan
L Learning	Learning from the experience and improving your ability to adopt new technologies in the future

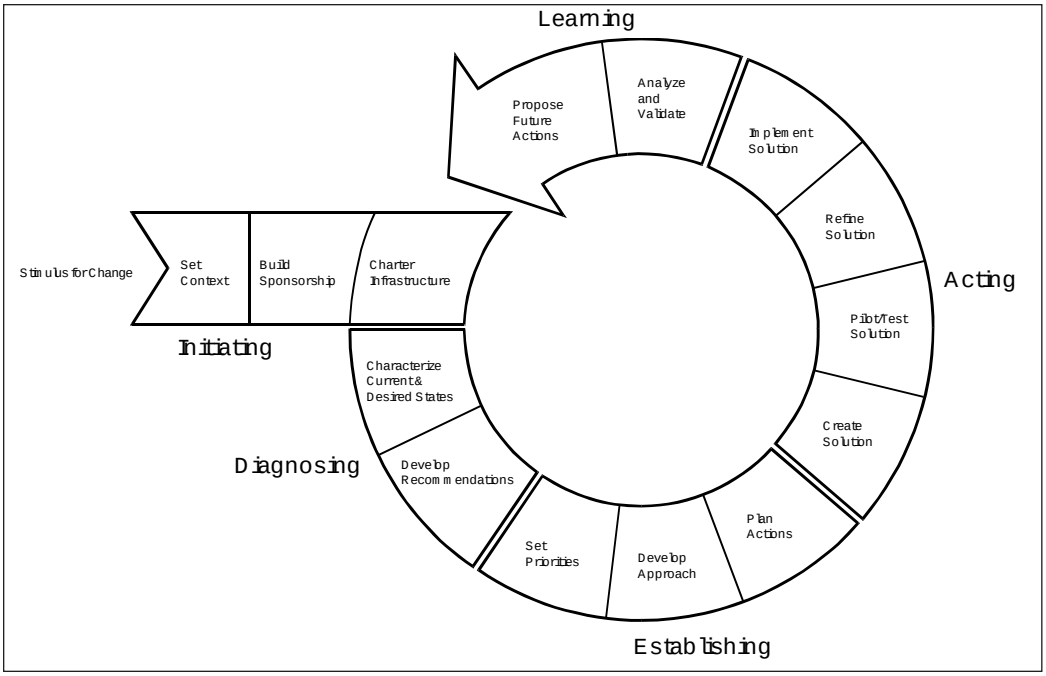


Figure 1: *The IDEAL Model*

restricted in terms of breadth and depth. In terms of breadth, the language used was process-specific; in terms of depth, it was seen as a life cycle model for larger, long-term efforts, such as moving an organization from one maturity level to another. The potential for extending the model's application to other technology domains and to efforts of virtually any size soon became clear, and the SEI launched an effort to reframe the model in a way that would facilitate its broader application. This effort resulted in the representation shown as Figure 1.

The more generic language used also opened the door for adopting the IDEAL Model as a standard life cycle that can be applied at all levels of activity within any major change effort (for example, in SPI terms, a maturity level, a key process area, a key process area component, and an activity).[4] Some find the multiple levels of applicability helpful.

In the learning phase of IDEAL, the adoption or improvement experience is reviewed to determine what was accomplished, whether the effort met the intended goals, and how the organization can more effectively or efficiently implement change in the future. Addressing any and all of these concerns represents active or double-loop learning.

As a whole, the model reinforces learning through the concept of continuous process improvement, but clearly IDEAL locates specific learning activities in the learning phase. Some have observed that learning should not reside in one phase (as after-the-fact reflection). Instead, learning needs to be recognized as an interwoven process thread. However, threading has not been expressly called out in the existing model.

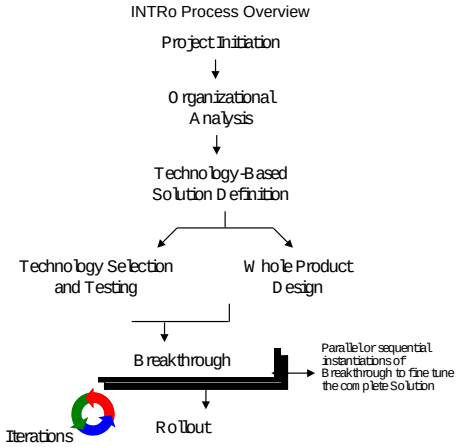


Figure 2: *IDEAL-Based New Technology Rollout*

INTRo

INTRo is a web-based process guide focused on making connections among business problems, value propositions, technology solutions, and their implementation.[5] INTRo was designed to help organizations adopt and implement new tools or technology. Since many such efforts are complex, and the effects so far reaching, a structured approach is required. Successful technology change management also requires comprehensive knowledge and skills that often do not reside in a single organization, individual, or team. Typical practitioners do not have the full range of skills needed to manage technology adoption. INTRo helps to fill the skill gap.

All approaches to technology change involve assumptions and expectations about what is central in a change effort. Many see "new technologies dominating the improvement effort; others focus on changes in process or business practices; some perceive organizational change. Very few, however, perceive change in multiple facets of the organization (process, technology, people, culture, organization); they commonly recognize change in no more than one or two dimensions [10]. INTRo was designed to recognize the multiple dimensions of change, including multiple subsystems in the organizational system, and to integrate these perspectives.

IDEAL provides a usable, understandable approach to continuing improvement by outlining the steps necessary to establish a successful improvement program. INTRo goes further, embodying the detailed how-to information needed to manage the introduction of a new technology, organized into a work breakdown structure of stages, steps, and tasks. Tips, checklists, guidelines, and tutorials accompany process descriptions. Related work products, such as a project schedule, a configuration baseline, and a training strategy, are also included in template form.

INTRo consists of seven stages, as summarized in Table 2.

Learning in INTRo

INTRo departs structurally from IDEAL and learning is not treated separately. Tasks that facilitate knowledge integration are interwoven. Learning is built into the process, rather than accounted for at the end in a lessons learned or post-mortem activity. The approach to knowledge integration is accomplished a number ways, and each of these is discussed briefly.

- stage management and stage end assessment activities
- process and product improvement activities
- knowledge and skills transfer mechanisms and strategy
- just in time: techniques, tutorials, related kernals
- process threads
- coordinating mechanisms

Stage Management, Stage-End Assessment Activities

After project initiation, all of the stages begin with a stage management step and end with a stage-end assessment step. Stage management sets the stage and ensures the readiness to begin, with tasks that include kickoff, monitoring of project progress, issue identification and resolution, and management of exceptions. Stage-end assessment, at the end of a stage, involves reviewing and baselining project/deliverables and preparing for the next stage.

Process and Product Improvement

Stage-end assessments sometimes include process review activities. These reviews verify that all planned processes have been completed according to the standards of the stage. Based on the results, problem reports or change requests may be generated—on the process being audited—to initiate process improvement activities. In rollout, the final stage of INTRo, extensive product and process improvement activities are performed. These tasks involve review and baseline of project/deliverables, collection of project feedback and metrics, metrics analysis, analysis of product quality, and completion of process reviews.

Knowledge and Skills Transfer Mechanisms and Strategy

Key activities highlight learning. For example, knowledge and skills transfer focuses on determining the strategy, including roles and responsibilities, and mechanisms for knowledge transfer and support of the new technology after rollout when in maintenance mode. The strategy covers mechanisms that will take

Stages and Steps	Tasks: Key Activities
Project Initiation Project Kickoff Project Organization Product/object Management Procedures Scheduling, Budget, and Control	§ Identify business drivers and objectives; Define scope; develop overall approach § Establish team; Determine costs § Identify key products, Define configuration management, product reviews, product baselines, reuse strategy § Develop schedule; produce budget; establish project control procedures; Perform cost/benefit and risk analysis
Organizational Analysis Requirements Definition Current State Baseline Desired State	§ Define high-level requirements; Collect previous change effort information; Draft high-level sketch of the solution § Review existing processes and assets; Develop a business process model of the current state; Review technical architecture requirements; Gather data about culture § Create list of improvements; Investigate technology ideas; Design new business process model; Identify levers of change, Identify sources of resistance
Technology-Based Solution Definition Technology-Based Solutions/Options Approach for Breakthrough and Rollout	§ Package solution options; Perform analysis on multiple solutions; Make recommendations; Determine increments for solution; Develop plan for acquisition; Identify products for evaluation § Develop high-level plan for Breakthrough; Determine iterations for breakthrough and rollout; Develop resistance mitigation strategy; Conduct stakeholder briefings
Technology Selection and Testing Architecture/Components Identification & Selection Technology Procurement Architecture Installation & Testing New Technology Test Scenarios	§ Evaluate and select the technical infrastructure; Include performance testing § Review contract agreements; Obtain required signatures; Generate purchase orders § Install components for testing; Setup user access rights; Test multi-user access and component configurations § Review test results and baseline architecture
Whole Product Design Social Design, Policies & Standards, Support Mechanisms, Training Preparation Knowledge Skills & Transfer Mechanisms	§ Review resistance mitigation strategy; Revise reward, incentive, and compensation programs; Restructure organization if needed § Establish policies to support the technology; define standards § Determine help desk & technical support; Develop documentation; Determine communication mechanisms § Identify training requirements for all roles/levels; Develop training strategy; Design or produce training materials; Pilot the training on those who will support the new technology § Knowledge & skills transfer to ensure success after rollout; Identify user group members and develop charter for group
Breakthrough Breakthrough Kickoff Installation & Solution Testing Breakthrough Monitoring	§ Detail breakthrough plan; Create post installation test scripts; Conduct training to breakthrough team § Install infrastructure components; Customize or configure components; Migrate data; Perform acceptance testing; Test whole-product solution § Monitor and manage resistance; Review solution testing results; Implement critical changes; Schedule other changes before rollout begins
Rollout Iterative Rollout Planning Rollout Launch Installation & Customization Iterative Rollout Review	§ Detail the plan for rollout; Create post installation test scripts § Conduct briefing with rollout users; Conduct training § Install the infrastructure components; Customize/configure the technology; Migrate data; Perform acceptance testing; Enable support mechanisms & other whole-product solution elements § Monitor and manage resistance; Review effectiveness of solution; Implement critical Change Requests and schedule remaining CRs before next rollout § Baseline business process and technical architecture models; Review metrics and make recommendations for process improvement; Make product improvement recommendations

Table 2: *Stage, Step and Task Summaries for INTRo*

the metrics analysis data, knowledge gathered from technical support, and help-desk personnel into account. A user group for process feedback is created. Information flow is planned among this group, technical support, and the help-desk team.

Just in Time Learning: Techniques, Tutorials, Related Kernels

INTRo offers techniques and related kernels of information, in context, in the web-based guide. For example, the requirements definition step includes a technique on critical requirements analysis; the current state baseline step includes techniques on use case modeling and business process modeling. Other techniques are provided for performing personal and walkthrough reviews, etc.[6]

Process Threads

INTRo takes a first step in building process threads for tasks that are related but conducted at different points in the process. For example, early in the organizational analysis stage, we begin a thread related to organizational culture. Cultural analysis is important: the information gathered will tell about previous change efforts that were more and less successful. The organization's culture also will influence our approach for breakthrough and rollout. The thread is woven into subsequent stages, through development of the resistance mitigation strategy, and the design of the whole-product solution. Process threading allows us to build on previous learning in a topic/area and emphasizes the iterative nature of solution development. The so-called answer on the culture of an organization is not gained in a single task but through a series of relevant, related activities performed over time. Initial activities in culture process thread are summarized in Table 3.

Coordinating Mechanisms

These make up the engine for knowledge management. All of the learning features already described support coordination and integration. These mechanisms relate to both the structure and content of INTRo, and range from

- activities, such as stage management and stage-end assessment; process and product reviews
- associated activities: use of process threads
- artifacts: knowledge and transfer strategy, resistance mitigation strategy, solution definition, whole-product design, etc.
- agents: project team, breakthrough teams, user groups, learning center

Still Learning

INTRo is undergoing end-to-end testing and the development team is gathering comments and requirements from reviews and pilot testing; the next version of INTRo will incorporate this input. Version 1.0 established a baseline that may now be refined and enhanced.

Colleagues working in different cities and in different organizations co-developed INTRo. Organizations adopting new technology also often have the need to collaborate among distributed locations—INTRo requires the use of collaborative techniques. We used collaborative tools to enable our work, and as we were developing, we captured data on how the remote collaboration was working. We are analyzing this data and may encapsulate practices to be included in future versions of INTRo that support cooperative work on virtual or distributed teams.

In many ways, INTRo represents an intersection among process management, knowledge management, and collaboration tools and techniques—all crucial aspects of organizational improvement and organizational learning. Future versions will make more deliberate connections among these disciplines.

Organizational Analysis Requirements Definition Current State Baseline Desired State	- Collect previous change effort information - Gather data about culture - Identify levers of change, Identify sources of resistance

Table 3: *Initial Activities in Culture Process Thread*

Conclusion

In part one, Integrating Knowledge and Processes in the Learning Organization, we showed that local adjustments were necessary within the process and knowledge-creation movements. Within the process arena, it remains for us to balance process formalization with process creation by leveraging individual knowledge and diverse perspectives through information exchange. Within the knowledge-creation arena, the challenge to filter and channel information for decision-making awaits. Organizations that support information sharing and knowledge creation are much more likely to establish effective and efficient processes and to improve organizational life [11,12].

In practice, technology change management represents the fusion of technology innovation and process and knowledge management as it is fully defined, operationalized, and enacted in a learning organization [13]. In part two, we looked at frameworks and tools that pull together process, knowledge management, and technology to support learning and successful change. We illustrated this with the IDEAL℠ model for software process improvement, and INTRo, a web-based process guide for technology change management. To different degrees, and levels of detail, both support the premises of active learning. INTRo places greater emphasis on organizational learning and enabling technology.

As we approach the 21st century, it is clear that our understanding of organizations, learning, and work is still unclear, changing, and likely to keep changing. We have yet to envision the future of organizations—as adaptive, virtual networks of activity. If we accept this challenge and the shift that it represents, we will begin to effect solutions that reflect this vision and foster communication, coordination, and collaboration. As our experience base grows, so will our ability to create spaces where active and interactive learning routinely occur.◆

About the Author

Linda Levine leads the effort at SEI on IDEAL℠ transition framework development, aimed at extending improvement models into structured processes for tech-

nology adoption and rollout. She is a process developer for INTRo. She also researches technology suppression, reasoning and communication, design disciplines, and the relationships between organizational learning and the use of collaboration technology. Dr. Levine hs her doctorate from Carnegie Mellon. She publishes widely and is the co-founder of the Working Group (8.6) on Diffusion, Transfer, and Implementation of Information Technology, part of the federation for Information Technology for Information Technology (IFIP).

Software Engineering Institute
Carnegie Mellon University,
Pittsburgh, Pa., 15213
Voice: Tel. 412-268-3893
Fax: 412-268-5758
E-mail: ll@sei.cmu.edu

References

1. Wellins, Richard S. and Murphy, Julie Schulz. (1995). Re-engineering: Plug into the Human Factor. *Training and Development*, pp. 49, 33-37.
2. Kock, Ned. (1999). *Process improvement and organizational learning: The role of collaboration technologies.* Hershey, Pa.: Idea Group Publishing.
3. Levine, L. (1997). An ecology of resistance. In T. McMaster, E. Mumford, E. B. Swanson, B. Warboys, and D. Wastell (Eds.). Facilitating Technology Transfer Through Partnership: Learning from Practice and Research. IFIP TC8 WG8.6 *International Working Conference on Diffusion, Adoption and Implementation of Information Technology*, pp. 163-174. Ambleside, Cumbria UK, London: Chapman & Hall.
4. Schein, E. (1996). Three Cultures of Management: The Key to Organizational Learning, *Sloan Management Review*, Fall, pp. 9-20.
5. Christie, A.M..; Levine, L.; Morris, E. J.; Zubrow, D.; Belton, T.; Proctor, L.; Cordelle, D., and Ferotin, J-E. (1996). Software process automation: Experiences from the trenches. (*SEI Technical Report* SEI-96-TR-013). Pittsburgh, Pa.: Software Engineering Institute.
6. Grudin, J. (1995). Groupware and cooperative work: Problems and prospects. In R.M. Baecker, J. Grudin, W.A.S. Buxton and S. Greenberg, (Eds.), *Readings in Human Computer Interaction: Toward the Year 2000* (pp. 97-105). San Mateo, Calif.: Morgan Kauffman.
7. Skinner, B. F. (1948). *Walden Two.* New York: The Macmillan Co.
8. Argyris, C., Good Communication that Blocks Learning, *Harvard Business Review*, July-August 1994
9. Gremba, J. and Myers, C. (1997). *The IDEAL℠ Model: A practical guide for improvement*. Bridge, Pittsburgh, Pa.: Software Engineering Institute. Also at www.sei.cmu.edu/ideal/ideal.bridge.html
10. Price Waterhouse Change Integration Team. (1995). *Better change: Best practices for transforming your organization.* Chicago: Irwin Professional Publishing.
11. Brown, J. S. and Duguid, P. (1991). Organizational learning and communities of practice: Toward a unified view of working, learning, and innovation. *Organization Science 2(1)*, 58-82
12. Brown, J.S. and Gray, E.S. (1995). After reengineering: The people are the company. *Fast Company, Premiere Issue*, pp. 78-81.
13. Lundberg, C. C. (1991). Creating and Managing a Vanguard Organization: Design and Human Resources Lessons from Jossey-Bass. *Human Resource Management*, 30(1), 89-112.

Acknowledgements

Recent writing on INTRo done with Geralyn Syzdek (Computer Associates) and Eileen Forrester (SEI) significantly influenced this article. I would like to express my appreciation.

Notes

1. The Software Engineering Institute is a federally funded research and development center sponsored by the Department of Defense. IDEAL℠ is a service mark of Carnegie Mellon University. For more information visit www.sci.cmu.edu/IDEAL.html
2. INTRo is a collaborative effort between SEI and Computer Associates (formerly Platinum technology, Inc).
3. For more of Argyris' insight, refer to Argyris, C. (1976). Single- and double-loop models in research on decision making. *Administrative Science Quarterly*, 21(3), 363-375. ; and Argyris, C. and Schon, D. A. (1996). *Organizational Learning II*, Reading, Mass.. Addison-Wesley, et. al.
4. IDEAL is similar to other improvement models, (e.g PDCA) but differs in subtle and important ways (Myers, 1998). PDCA is an acronym for Plan, Do, Check, Act: four steps in an improvement cycle that is widely used in Total Quality Management (TQM). This cycle is described by W. Edwards Deming's *Out of the Crisis*. Deming is well known as the father of PDCA.
5. INTRo is a co-development effort between the SEI and Computer Assoc. (formerly Platinum technology Inc.)
6. These techniques and related kernals are made available by Computer Associates (formerly Platinum technology Inc.) as part of the INTRo co-development effort. They are process assets in the Process Library in Platinum's Process Engineer tool set.

Suggested Additional Readings

Bannon, L. J. and Kuutti, K. (1996).Shifting perspectives on organizational memory: From storage to active remembering. In J. Nunmaker and R. Sprague (Eds.), *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*, (pp. 156-167). Maui, Hawaii: IEEE.

Roth, G. L. (1997). Learning Histories: Using Documentation to Assess and Facilitate Organizational Learning, Working paper, MIT Sloan School of Management. At http://learning.mit.edu/res/wp/index.html

Schein, E. (1995). Kurt Lewin's Change Theory in the Field and in the classroom: Notes Toward a Model of Managed Learning. Working paper, MIT Sloan School of Management. On the web at http://learning.mit.edu/res/wp/index.html

Schein, E. (1995). "On Dialogue, Culture , and Organizational Learning," E.M.R. Spring, pp. 23-29. (reprinted from *Organizational Dynamics*, Autumn/1993)

Schein, E. (1997). Organizational Learning: What is New? Working paper, MIT Sloan School of Management. On the web at http://learning.mit.edu/res/wp/index.html

Senge, P. (1990). *The Fifth Discipline.* N.Y.: Doubleday.

Lessons-Learned Web Sites

http://afkm.wpafb.af.mil/ASPs/Def_SubjArea(1).asp
Subject areas in the Air Force Knowledge Management index cover operations (flying ops, space, intelligence, missile, C2, etc.), logistics, support (security, personnel, etc.), medical, professional, standardization (scientific research and development, acquisition, contracting, etc.), special investigations, history, and command policy.

http://call.army.mil/call.html
Center for Army Lessons Learned virtual research library. This page contains one or more hyperlinks to resources external to the Department of Defense. These hyperlinks do not constitute endorsement by the Center for Army Lessons Learned.

http://www.safaq.hq.af.mil/acq_ref/bolts/bolt10/lb10_team/final_report/acqpla15.html
Lessons Learned and the Navy "Turbo Streamliner." The Navy has developed an electronic compendium of lessons learned in applying acquisition reform which may be particularly helpful in reviewing requests for proposal, but also provides valuable information to request for proposal writers. The Navy's program, an interactive tool called "turbo streamliner," was the product of a centralized request for proposal review team established to assess how well acquisition reform has been incorporated into Acquisition Category I requests for proposals.

http://llis.nasa.gov/
The purpose of the NASA Lessons Learned Information System is to collect and make available for use the lessons learned from almost 40 years in the aeronautics and space business. Access is restricted to NASA, approved NASA contractors, and other approved United States government organizations. Use requires either Netscape 4.0 or higher and Internet Explorer 4.0 or higher, and activating the browser's Java capabilities. Only a Java-capable browser will be able to navigate the site fully.

http://joy.gsfc.nasa.gov/modsd/lessons.html
The Mission Operations and Data Systems Directorate has developed a broad and exhaustive wealth of knowledge about ground data systems based on many years of experience engineering, developing, integrating, maintaining, and operating them. Key information is captured in a lessons learned database Reusable Experience with Case-Based Reasoning for Automating Lessons Learned.

http://www.cnsl.spear.navy.mil/Y2K/cnsly2k/support-faq/lessons.htm
Army and Navy Year 2000 Lessons Learned

http://ryker.eh.doe.gov/ll/sells/faq.html
The Society for Effective Lessons Learned Sharing develops fact sheets to help lessons learned professionals implement and improve lessons learned programs. Subject areas include, but are not limited to, a lessons learned list server, tips for writing lessons learned documents, corrective action, and program review criteria.

For a humorous look at Lessons Learned . . .
http://top7business.com/archives/1998/09/090998.html
Top 7 Lessons Learned While Growing Up on the Farm

http://members.tripod.com/~musone/learned.html
One Man's Look at Personal Lessons Learned

Off-the-Shelf Software: Practical Evaluation

Lee Fischman and Karen McRitchie
Galorath Incorporated, The SEER Product Developers

This article provides practical advice on evaluating off-the-shelf software by exploring the constellation of success factors common to all such software, from stand-alone applications to low-level components. The article attempts to reduce much of what has been discussed in previous literature into a few succinct sections. The factors that we develop are particularly suited to evaluating contractor proposals for utilizing off-the-shelf solutions.

With defense and commercial technology so tightly interwoven, developers must evaluate commercial and other off-the-shelf[1] options—and a client must evaluate the developer's proposed solution—in practically every project undertaken. An evaluation is best carried out by engaging in a cost-benefit analysis: costs and risks of implementation vs. benefits delivered. This paper presents the criteria for this analysis.

No Free Lunch

Off-the-shelf software may offer tremendous benefits but there is no such thing as a free lunch. Any department manager forced to support desktop software and locked into an endless upgrade cycle can tell you that.[2] The essential drawback to off-the-shelf solutions is that they do not allow absolute control over the origin and often the ongoing baseline of code upon which they rely.

These are a few of the cautions that go with using off-the-shelf software:

Items are often unique. Uniqueness demands that developers adapt to new ideas and methods, which can be time-consuming and costly.

The efficacy of stand-alone applications is debatable. Advertising does not always translate into reality, and this may lead to costly technical resets.

New paradigms may be unwelcome. Off-the-shelf technologies may satisfy user needs, but may be incompatible with current doctrine.

Implementation may not be rigorous enough. An off-the-shelf item may not be developed to the performance standards required in some applications.

Coverage may only be partial. Off-the-shelf software may not be intended specifically for the target application and so may offer only partial solutions.

Vendor support may be critically unavailable. Outside developers are not directly under your control and are often not available when most needed.

*Products may be volatil*e. Volatility can increase maintenance costs as technical-refresh requirements are increased. As support for a baseline technology disappears, advanced obsolescence may be the result.

Do not let everything above scare you, for off-the-shelf software still is fundamentally a good thing. The task is to maximize the return on its use while minimizing potential drawbacks.

Off What Shelf?

The many flavors of "off-the-shelf" range from fully commercial-off-the-shelf (COTS) to simply reused code. Our classification scheme is provided as a framework for thinking about the success factors that follow, and also to dispel the notion of COTS as a monolithic entity. Everything in Chart 1 can be COTS.

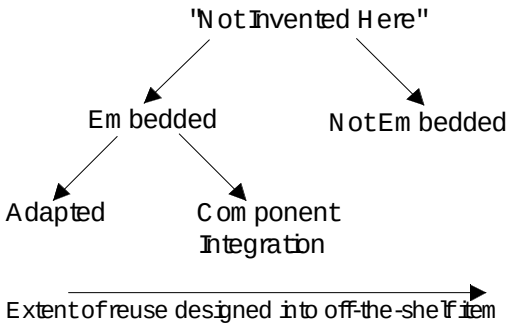


Chart 1 *Framework for considering COTS success factors*

Nonembedded items are essentially stand-alone applications, while embedded items undergo tight integration directly into delivered systems. The latter are further divided into adapted software (old code, designs, modified applications) vs. components intended for reuse (programming libraries, application program interfaces, stand-alone code such as DLLs). As Chart 2 shows, the choice between using embedded and nonembedded items partially rests with how specialized the application is.

Integration Factors

To get an off-the-shelf item ready for use, you have to integrate it. Integration is the cognition and effort required to get something you did not create to work properly. This can be as simple as installing something like a commercial word processor on your desktop, or as complex as the big-team, multiyear effort required for integrating enterprise resource planning software.

The following sections cover the interrelated components of an off-the-shelf implementation analysis: criticality, scope, and "meta" and "micro" success factors. Analysis components must be linked to assess the significance, risk and prospects for fulfilling integration (See Table 1).

		Effort	Fulfillment of Project Goals	Risk to Overall Project
Scope	Size of the job	X		
Criticality	Criticality of integration effort to overall application requirements			X
"Meta" Factors	High-level indications of success		X	X
"Micro" Factors	Low-level indications of integration effort	X		

Table 1 *Assessing significance, risk, integration prospects*

Criticality

If an integration effort fails, what must be done to insure the entire project is not jeopardized? Any management approach with (the hope of) strong risk controls must extend its realm into off-the-shelf issues as well:

- Do alternate off-the-shelf options exist as backups?
- Has an up-front evaluation insured that failure will not occur?
- If this off-the-shelf integration effort fails, can the project stand to lose the functionality?

Scope

Integration factors are only useful when the scope of the integration effort is understood. For example, a large project with a small integration task does not face much risk due to integration uncertainties. We present two scoping methods depending on the amount of knowledge you have:

Quick Sizing—Although an imprecise method, quick sizing does not require much information and it can be used either with code-level or "black box" integration. The idea is to establish the magnitude of off-the-shelf items by deciding whether (by fuzzy analogy) they simply are small, medium, or large. This magnitude is then adjusted by the proportion of the application that will be used. Table 2 illustrates the process:

Attribute Sizing—This method yields fairly precise indications of size, although its use is limited to code-level integration. A size metric of some sort is used to quantify the off-the-shelf software; we recommend the number of functions and data tables actually being used. If the software is object-oriented, its number of parent classes and methods across all instantiations might be counted. Alternate metrics could probably be substituted with similar intent, although correlation with effort needs to be determined.

"Meta" Factors

The factors laid out in Table 3 are most appropriate for high-level evaluation of reuse plans,[4] particularly when compared against column entries representing key management objectives. A client or program office could use these as a scoring chart to gauge the relative merit and risk of contractor proposals for integrating off-the-shelf solutions. We have provided our assessment of factor impacts with arrows indicating positive or negative effect; darker arrows represent a more definite effect.

There is not enough space to discuss how all row weightings in Table 3 were derived, but we will discuss the process behind "Maturity/Stability of Product" to give an idea of the thought process involved.

Development Schedule—The less mature a product, the greater the frequency of more significant upgrades. Waiting for upgrades could introduce development lags.

Project Risk—A more mature, stable product reduces risk by having a well-understood COTS baseline to work with.

Purchase Price—A less mature product could cost less to induce purchases, but it could cost more because the developer needs to recoup costs.

Development Cost—Lower maturity implies increased component volatility, and changes will take time for developers to learn.

Maintenance Cost—A mature product will refresh less and require less ongoing effort to incorporate new versions.

Quality—The likelihood of undetected defects greatly increases with a less mature product, therefore, maturity will increase delivered quality.

As you can see, the weighting process is quite intuitive; comparative weights are recommended instead of absolute, numerical ones. Weightings can vary significantly depending on the component and project, so we recommend you re-evaluate weights as necessary. Evaluate off-the-shelf options against a custom-developed one, if that is the alternative.

"Micro" Factors

These factors are intended to address detailed integration issues involving off-the-shelf code. A developer could use these to gauge reuse risks and potential costs at a highly detailed (perhaps component) level.

OFF-THE-SHELF PRODUCT CHARACTERISTICS

Component Type—The level of access provided, and required. Ranges from pre-existing code requiring extensive modification to pre-built, encapsulated components with discrete interfaces.

Component Volatility—How mature is the product, how often will it be upgraded, and to what extent? Ranges from an alpha release to a highly mature product.

Component Application Complexity—The difficulty of the application, in terms of the amount of study and experience required to become proficient in using it. Ranges from a simple, general application to one that is very peculiar and difficult to fully understand.

Interface Complexity—The design and coding overhead

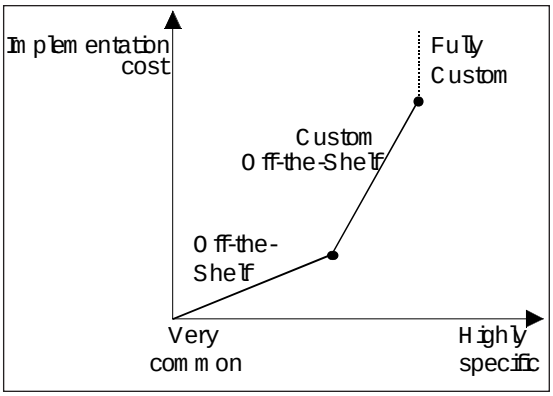


Chart 2[3] *Degrees of application specialization*

Table 2 *Quick-sizing magnitude of off-the-shelf items*

Size of Off-the-Shelf Software	Examples	Extent of Functionality That Is Used		
		Some (< 40%)	Much (30 – 70%)	Most (> 60%)
Small	Minor set of functions that deliver functionality at a subsystem level with no further methodological ramifications; minor application (text editing, etc.) to be incorporated into target system.	Very Small	Small	Small
Medium	Major set of functions that require a "methodology shift"; major subsystem, such as a database engine or graphics rendering.	Small	Medium	Medium
Large	Major library that is fundamental to proper exploitation of the target environment; primary application that will be modified to current requirements, such as desktop software or highly application-specific software.	Medium	Large	Large

required to integrate this component. Ranges from a simple, stable interface to one that incorporates many factors that change with the circumstances of implementation.

Product Support—The degree of access to vendor or effective third-party support. Support may be as good as an onsite consultant or as nonexistent as a disbanded original development team.

INTEGRATION INSTANCE

Component Selection Completion—The extent to which the off-the-shelf component has been identified and evaluated. Ranges from selection completed to no product evaluations done.

Experience With Component—The developers' experience with the component: Is each integration of this a significantly new and complex task, or has it become a standard operation?

Learning Rate—Rate at which people can learn while implementing a component. Ranges from exceptional opportunities for learning-while-doing to a massive task with very little opportunity for learning-while-doing, with each integration being new territory.

Reverse Engineering—The percentage of component functionality that must undergo thorough review by technical staff. Ranges from "black box" to line-by-line review and testing.

Component Integrate and Test—The integration effort required for this component; rank this against what typically is required to integrate a home-grown unit of code into a software program. Ranges from above-average, intimate dependencies that need to be addressed to virtually no effort to be integrated into the target system.

Test Level—The rigor and formality of testing is related to contractual requirements and the potential for loss if the software malfunctions during operation. Ranges from "highest reliability" or "public safety" (rigorous testing, following prescribed plans, procedures, and reporting) to "slight inconvenience" (minimal testing, no prescribed procedures or reporting).

For a risk analysis, these micro-factors can be ranked using the suggested ranges, compiled with a weighting scheme, and overall scores can be compared for differ-

	Notes	Development Schedule	Project Risk	Off-the-shelf Purchase Price	Development & Integration Cost	Maintenance Cost	Quality
Target project/environment							
Openness of architecture	Highly standardized, able to accomodate wealth of prebuilt components under stable, well-understood conditions	*i*	***i***	*i*	***i***	***i***	*h*
Flexibility of requirements	An envelope, rather than a rigid benchmark, within which an integrated item's performance and function can lie	***i***	***i***	*i*	***i***		***h***
Attitude of staff	Progressive staff with a desire to balance what is new and good with a fair degree of caution	*i*	***i***	*i*	*i*		*h*
Evolutionary emphasis in system requirements	A system architecture and interfaces that are robust enough to accomodate significant product upgrades		***i***	*i*		***i***	***h***
Off-the-shelf product and provider							
Maturity/stability of product	Proven over time and in the marketplace	*i*	***i***		***i***	***i***	***h***
Commercial competitiveness of product	A well-developed market with numerous suppliers		***i***	***i***			***h***
Supplier "reputation"	Ability to support and respond over the long term		***i***	*h*		***i***	***h***
Evolutionary emphasis in product strategy	Version changes are not radical, and are inclusive of the existing customer base		***i***			***i***	***h***
Transparency	Insight into—and ability to modify if necessary—internals of code	***i***	***i***	*h*	*i*	***i***	***h***
Refresh Frequency	Pace with which new versions are released, and extent of modification	***h***	***h***		*h*	***h***	
Functional match	Appropriateness of off-the-shelf item to the current application	***i***	***i***		***i***	***i***	***h***
Performance suitability	Suitability of off-the-shelf item to application's performance requirements	***i***	***i***		***i***	***i***	***h***

Table 3 *Factors for high-level evaluation of reuse plans*

ent components within a project. We also find that component integration is somewhat analogous to other kinds of software reuse, and we have mapped these factors into cost determinants of reuse effort to obtain a rough integration cost model.

Running the Cost-Benefit Analysis

There are two kinds of off-the-shelf vs. custom comparisons**,** easy ones and hard ones. It is not worth dwelling on the easy comparisons, such as whether to build something like a word processor from scratch or buy one. However, the more difficult choices will need a cost-benefit analysis. Costs and benefits may not always be expressed in dollar amounts; if this is the case, then you must resort to relative comparisons.

Obtain dollar costs for custom development using any suitable method, including an estimating model. Off-the-shelf costs, meanwhile, include purchase and vendor support fees plus actual integration effort; a small number of cost models are available to estimate the latter. The nonmonetary costs of integration have been covered with the micro- and meta-factors presented here. Make certain to apply these factors to both off-the-shelf and custom development options.

To a great extent, software benefits are accounted for in terms of functionality delivered. Off-the-shelf software may offer a substantial solution but not a complete one, at least not what a custom product could deliver. User flexibility therefore goes a long way towards certifying that something not homegrown is good enough. For example, can performance requirements be relaxed? If no single "silver bullet" exists then a fine-grained amalgam of solutions may approach a full solution for the target application.

Make certain that benefit evaluation extends into the hidden realms beyond mere functionality delivered. An off-the-shelf option may offer better vendor support, less expensive service, and plentiful expansion options. Other advantages can be harder to quantify, such as ease of use and a community of like- users. Significant hidden benefits should be carried through into final comparisons.

You are going to have to mix dollar comparisons with less tangible costs and benefits. Zero in on the factors driving your situation and let these guide your analysis, keeping the number of compared items to a manageable amount. This sort of multivariate comparison process is an ideal candidate for a method known as the analytic hierarchy process,[5] which basically provides a formal framework for the mixed comparison of many unrelated factors.

Summary

In planning off-the-shelf integration, some useful general management advice may be available, but it is really up to your careful research, preparation, and skills to make any effort a success. The next time you need to integrate off-the-shelf technology, the best tactic is to ask the supplier for a manual. If there is no manual, start worrying.◆

About the Authors

Lee Fischman is special projects manager at Galorath Inc. in El Segundo, Calif. He conducts research and development of SEER tools and consulting methods, and has explored software economics and estimating in numerous papers over the past several years.

Karen McRitchie is vice president of development at Galorath Inc. She has more than 10 years of experience in software and hardware cost estimating and reliability modeling. She has been a lead member of cost estimating teams on many major projects.

Galorath Incorporated, The SEER Product Developers
100 North Sepulveda, Suite 1801
El Segundo, Calif. 90245
Voice: 310-414-3222
Fax: 310-414-3220
http://www.galorath.com
Fischman's e-mail: fischman@galorath.com
McRitchie's e-mail: karenm@galorath.com

A list of related writings is provided at http://www.galorath.com/COTS_references.html

Notes

1. ***COTS is dead, long live COTS!*** From being an infrequent shortcut to the traditional build everything development emphasis, COTS has become pervasive in today's software development projects. The term is generic and can refer to many types of applications, levels of complexity and integration. Meanwhile, a host of other off-the-shelf solutions have matured, which, although not truly commercial, offer similar benefits. The COTS shorthand has become a kind of mantra, yet the more general challenge of using off-the-shelf software provides a much more complete lesson. We therefore define off-the-shelf software as any cogent code that is being reused.
2. Adequate vendor support frequently must be obtained through upgrades, as support is removed from older versions of software.
3. Chart 2 shows the relationship between specificity of requirements and implementation cost as a function of customization. As requirements grow more specialized, the cost of using off-the-shelf software increases. Past a certain point, off-the-shelf software must be mixed with custom work to fulfill specific requirements. The most specific needs (target recognition, radar cross-section calculation, etc.) may require fully custom solutions.
4. The emphasis in these factors was derived primarily from a literature survey. It is very hard to gather actual data; we are therefore interested in hearing from you. The chart has been posted on our web site at www.galorath.com/COTS_survey.htm
   You can suggest alterations to the factors, including alternate weightings, and your rationale for any changes you suggest. We will send the results of the survey to everyone who participates or simply registers.
5. A great deal of information on this method is available at www.expertchoice.com (no affiliation with Galorath Inc.).

Restoring Cyber Security

Bryan C. Crittenton
Vistronix Inc.

This article addresses the security management issues of millennium enterprise system environments. The intent is to define an approach that enhances cyber security and reduces negative system impacts through automated monitoring and management techniques. In addition, this article addresses the benefits of proactive security and systems management. This proactive approach details some of the automatic functional responses that can be initiated to alleviate a variety of security violations and systems malfunctions.

In the rush to provide Year 2000 (Y2K) compliance, many systems have not been adequately evaluated and validated from a security perspective. Most of the efforts have focused on date-oriented testing, with little concern for the post-Y2K security integrity of the system. Many of the patches and fixes developed have not been tested for potential security violations.

In the process of these developments, many security procedures have been bypassed to accommodate immediate access for online testing. Though some bypasses have been removed, many were left behind to allow for quick fixes. These backdoors may even be active in some commercial products. In short, the impact of the Y2K fixes may prove more devastating to the security of the operational systems than the Y2K bug itself.

The New Threat

Post-millennium systems and security administrators are facing several new complex problems. It is now obvious that Y2K impacts reach across the entire enterprise system to include security applications, security logs, security time stamps, and security access verification routines.

In addition to this wave of problems, several new security threats have appeared. New viruses are generated at an alarming rate, and some are designed to target the Y2K fixes. Some of the Y2K fixes have injected new bugs into operational systems and security management routines. In addition, several backdoors have been uncovered that are remnants of Y2K development efforts and test procedures.

Most fixes were never evaluated for security impacts or given sufficient operational time to ring-out all of the inherent problems. Many of the environment variables will not even be present for some time. This presents a staggering management challenge to all security and systems administrators.

Compromised Security

With the advent of these multiple challenges and simultaneous problems, current security practices will probably fail.

Security procedures that require manual intervention will be too slow to react before system damage occurs. Reactive systems and security management will not protect most enterprise systems in this new environment. Slow reactionary procedures may open the system to a high probability of swift and lethal damage. Over the next few months, all of the unknown variables will be brought into play. Most of the Y2K fixes have not been introduced to full battlefield conditions.

In addition, all the minor programming errors that have crept into these fixes can then become apparent. The full impact of various security violations, integration problems, interface disconnects, and programming errors will have to be dealt with in the near future. This collective impact is unpredictable. New and previously unknown bugs will appear, which will further challenge the security and integrity of our systems.

It must be recognized that many software developers have installed private access paths, or backdoors, in many of the online patches being implemented. Many Y2K development environments are such that none of the developers are certain as to how stable their patches or fixes are. Therefore, the implementation of backdoors has become a popular failsafe access for developers.

Unfortunately, the backdoors are insecure, and provide a vehicle for hackers to surf through the system. This new environment is a prime target for major security breaches.

System Monitoring and Management

The goal of Systems Monitoring and Management is to ensure that not only applications, but also all resources that support them, are online and performing optimally so user productivity and security remain high. Their availability, performance, and integrity are only as good as the system's weakest link.

Total systems management is best accomplished by taking a top-down data-centric rather than a bottom-up hardware-centric view. In the case of enterprise systems management, this means monitoring and managing the application, database, middleware, operating system, servers, network, and other related systems' hardware or software.

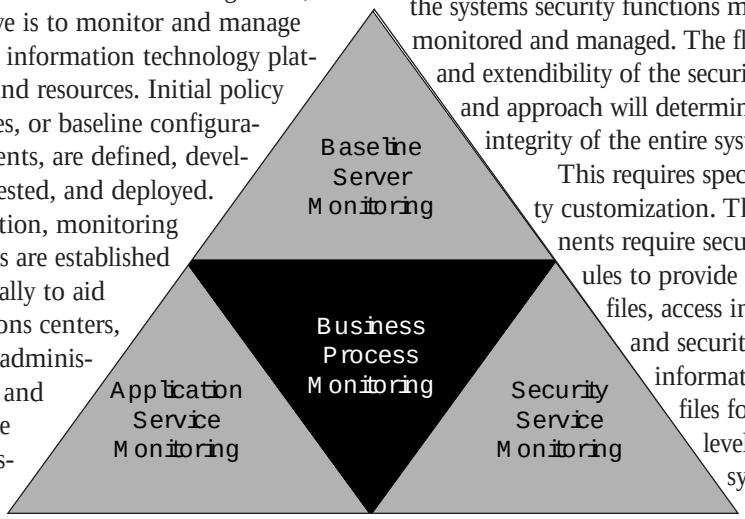
It is only when security and systems administrators view an entire environment, rather than a limited point-solution perspective, that they begin to support the overall objectives of true enterprise systems management: supporting end-user data access, security, and productivity.

Typically, multiple levels are involved in implementation of Enterprise Systems and Enterprise Security monitoring and management. Each successive level of implementation should add value to the previous level. In addition, security and systems administrators should assist in determining specific requirements for each level.

It is recommended that every system implementation include at least a baseline server monitoring function, an application service monitoring function, and a security service monitoring function. These areas of focus ensure that a significant level of operational protection is provided.

Capturing Enterprise System Data

One of the first levels of implementation is Baseline Server Monitoring. Here, the objective is to monitor and manage specific information technology platforms and resources. Initial policy baselines, or baseline configuration agents, are defined, developed, tested, and deployed. In addition, monitoring consoles are established specifically to aid operations centers, system administrators, and database administrators.



Necessary infrastructure and operations and administration support procedures also are established to support long-term stability. This level of implementation results in a fully operational, rapid deployment of baseline configured clients, agents, managers, and monitors. The baseline implementation establishes the foundation from which to implement the application service monitor or the security service monitor.

The next level of implementation is the application service monitor. Here, the management focus shifts from monitoring and managing individual systems to monitoring and managing the enterprise system. An internal service assurance manager and security service manager also should be implemented. At this level, flexibility and extendibility of approach and tools used are vital.

This phase generally requires specific customization to meet the unique systems or security requirements of the enterprise system topology. These components also wil require knowledge modules to provide product and platform information profiles on which tools can guide their management functions. When this level of implementation is reached, the approach must support the application service management, security management, and server monitoring and management.

The security level of implementation can be referred to as a security service monitor. Here, management focus shifts away from monitoring and managing application aspects of the enterprise systems to monitoring and managing the security of the enterprise system. At this level, all of the systems security functions must be monitored and managed. The flexibility and extendibility of the security levels and approach will determine the integrity of the entire system. This requires specific security customization. The components require security modules to provide user profiles, access information, and security level information profiles for multiple-level security systems.

Additionally, implementation should be oriented toward business process monitoring. This level of management service should be undertaken only after application service monitoring and security management are in place. At this level of implementation, the focus is to monitor and maintain critical day-to-day business processes. An example is monitoring the number of people using an application for license monitoring. When the number of users exceeds the license limit, the Business Process Monitor would generate an alert, and log out all inactive users.

This level of implementation adds business process monitoring to the functionality of the other previous levels.

Converting Data into Usable Information

By intelligently monitoring all of the necessary systems and security functional parameters, events, accesses, and processes, an effective management scenario can be implemented along with a proactive response agenda that will avoid major system malfunctions and security breaches.

The backbone of an automated management system is the data that is collected, and the response philosophy adopted.

Automatically monitoring all of the required functional parameters within the enterprise system allows the service monitors to build a baseline profile of the entire system. Once a baseline is established, trend information can be correlated.

With a trend database, a service monitor can identify out-of-tolerance conditions. In order to provide instantaneous responses, the service managers can automatically act upon these conditions, which can range from increasing security on endangered systems to putting failing devices offline, or calling the security administrator on a cellular phone to deliver a message about a breach condition.

Reactive vs. Proactive Management Environment

In a manually oriented management philosophy, viral attacks on the systems or security breaches could usually be dealt with in real-time increments. As systems have become more complex and communications more robust, reaction time has become more critical.

Management philosophies need to be reoriented to a more proactive posture. Problem responses now need to be handled on a nanosecond basis, which only a truly automated, proactive management approach can handle.

By implementing the new enterprise systems/security management approach, disaster can be avoided. For example, the automated trend analysis in the service monitor can indicate when a database caching size is incorrect. The service manager can then initiate a response function to increase or decrease cache size, eliminating a potential database failure.

A disk may be experiencing soft failures. The service monitor would detect this trend, and the service manager could soft-fail over to another disk drive or a backup copy of the data.

A security breach might be detected, and offending users piped to a dummy system while authorities are notified, all within nanoseconds.

Through the implementation of a security service monitor and a security service manager, all security functions can be dealt with at the computer's speed. Breaches can be contained, privacy can be maintained, new users initiated, inappropriate connections monitored or severed, virus handling automated, macros monitored, keywords tracked, and integrity verified. Security trend analysis information also could prove quite useful for future systems development.

Proactive Management Approach

A proactive management approach must be designed to manage exceptionally complex systems with high availability. Since most new applications demand global accessibility, constant availability is crucial.

Enterprise system applications may also be accessing and interacting with data in real time to support critical processes like tactical information processing. Availability requirements for these applications increase dramatically as users become more dependent on them.

The management approach must also be prepared for periodic and unscheduled disturbances, keeping the systems running regardless of a localized failure, or recover them quickly in spite of individual failures or breaches. For example, requests must not be lost if a processor or server goes down. This type of loss could result in lost information, increased operational cost, and lost confidence in the system.

Downtime for maintenance and unscheduled outages also can be planned and supported. Finally, the proactive management approach must be scalable to accommodate usual increases in load or multiple problems. As more applications come online, more concurrent usage can result in serious bottlenecks, loss of available services, lost incoming requests, and unacceptably slow response times. These are all taken into consideration in the proactive management system.

From a management standpoint, acceptable response times for applications must be no more than three seconds for typical actions, and no more than seven seconds for the longest operations. Failure to provide this rate of performance management can result in user dissatisfaction, and again, loss of system confidence.

For Internet applications, the performance management approach should expect larger-than-normal numbers of concurrent users. Managing Internet service demands that are above defined peak usage are not as predictable as similar demands placed on internal systems. As an example, a major news story could create an unexpected surge of hits to a site over a short time period. If any part of the enterprise system cannot handle the demand, cascade failures might occur, which may introduce security vulnerability.

Finally, the management approach must also manage all networked applications and the data they access. Both must remain secure to ensure absolute privacy and protection of sensitive information. Legal and financial exposures created by security breaches can be as high as tens of millions of dollars in damages per incident. With stakes this high, security concerns are critical.

Parameter Handling

The goal of enterprise management is to define the representation of management data, as well as the methods used to extract data or control managed objects.

The goal of security management is to define and enforce security policies, identify and channel the user community, and ensure that only authorized individuals access information. To meet diverse goals, parameter monitoring is designed to cover the entire technology stack from hardware drivers, to software applications, to network devices.

The data collected by parameter monitoring is translated into trend analysis databases and data models. New management technologies have been devised to support instrumentation of all systems and security applications and collections of parameter data required to quickly avert these systems problems.

To implement this technology, an enterprise systems management architecture was created. This architecture combines all the components to monitor, manage, and proactively recover from security breaches to systems malfunctions.

The enterprise systems management architecture has three basic components, as shown in the adjacent graphic.

This management architecture provides the definition of standard parameters, attributes, properties, and their association to the managed objects. The collection of the parameter data involves instrumentation of hardware, software, and access data, which is funneled to the service manager. This data can be used for advanced management, such as access and security correlation, security violation and problem diagnosis, or predictive availability modeling.

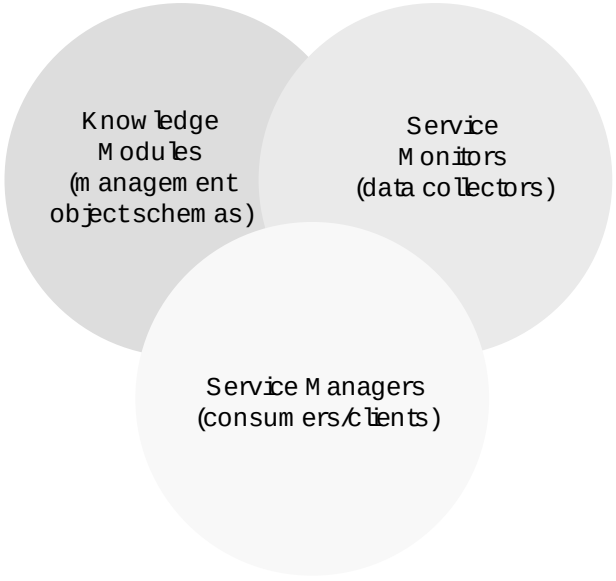
The resulting systems impact is increased security, availability, and reliability. Furthermore, this approach will reduce support personnel and systems downtime. The resulting operational cost reduction provides the driving force behind this new approach to enterprise systems and security management.

Conclusions

Systems and security administrators face multiple problems due to unknown and injected failures from Y2K solutions. These problems are multifaceted, parallel, designer bugs that could cause massive system failures and security breaches.

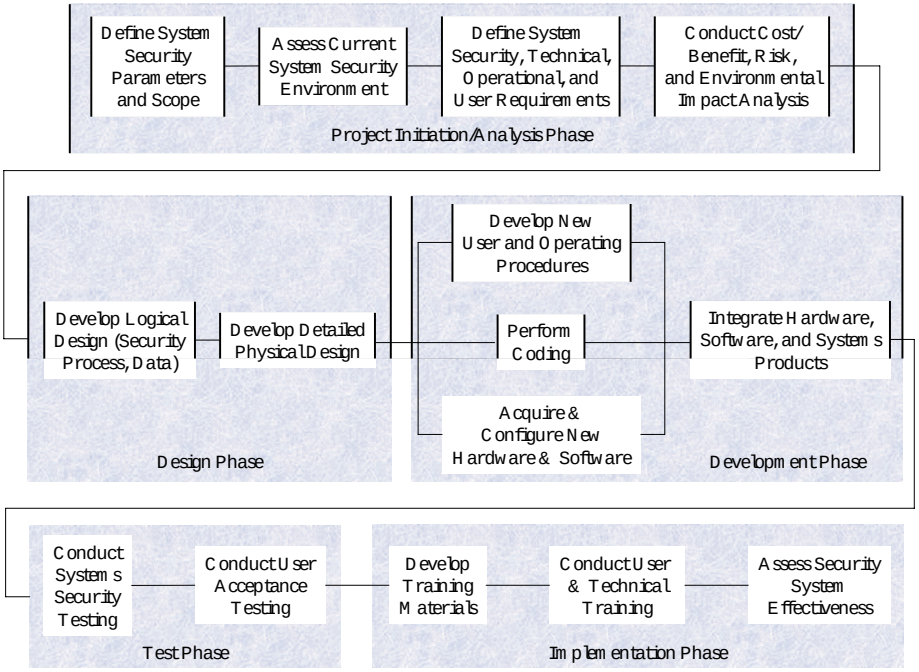
To counteract these looming failures, new automated management approaches must be implemented. These need to include well-thought-out automated systems responses that contain damage and protect system resources and users. Relying on existing manual procedures, or frameworks that do not provide a solid automated response to intrusions, viral attacks, or systems failures due to Y2K solutions will cripple most enterprise systems.

New enterprise management technology helps unify systems and security management. This approach to enterprise management provides a high degree of



conceptual breadth, and stands as an enabler for the next generation of systems and application management tools.◆

Our approach to enterprise systems and security management is depicted in the chart below:



About the Author

Bryan C. Crittenton is Director of Information Technology Programs for Vistronix Inc in Vienna, Va. He has more than 15 years experience supporting enterprise-wide management information systems in such areas as data, network, and Internet security and intrusion protection, network architecture analysis and design, department-wide financial and administrative systems design and development, and data warehousing and mining. Crittenton has numerous certifications in network design, software engineering, total quality management, and systems architecture and planning. He holds a master's degree in business administration from the Virginia Polytechnic Institute and State University and a bachelor's degree from the University of Virginia.

Vistronix Inc.
8391 Old Courthouse Road, Suite 205
Vienna, Va. 22182
Voice: 703-734-2270
Fax: 703-734-2271
E-mail: bcritten@vistronix.com

Web Addition

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

Content Change Management: Problems for Web Systems

Susan Dart
Dart Technology Strategies

Behind the facade of a web site lies the task of managing its infrastructure and content. This is driving the Internet economy into a web crisis. The software community has experienced a similar crisis and knows that configuration management (CM) is a key player in resolving it. Nine challenges facing web systems are presented. As the entire world becomes connected to the World Wide Web, content problems will be magnified. While traditional software CM provides a static solution (such as via a centralized development methodology creating batched, planned releases), content CM will provide a dynamic solution (via distributed, real-time updates) in response to user traffic monitoring. It is imperative that the lessons learned from CM are applied to web tools. Otherwise, the Web community is doomed to experience all the delivery, quality and complexity problems that have plagued the software community.

Web Addition

Content Change Management: Problems for Web Systems

Note: This is a reprint of an article original © Springer Verlag 1999,
from the proceedings of their 9th International Conference on SCM.

Susan Dart
Dart Technology Strategies

Behind the facade of a web site, lies the task of managing its infrastructure and content. This is driving the Internet economy into a Web crisis. The software community has experienced a similar crisis and knows that configuration management (CM) is a key tool in resolving it. Nine challenges facing web systems are presented. As the entire world becomes connected to the world wide web, content problems will be magnified. While traditional software CM provides a static solution (such as via a centralized development methodology creating batched, planned releases), content CM will provide a dynamic solution (via distributed, real-time updates) in response to user traffic monitoring. It is imperative that the lessons learned from CM be applied to web tools. Otherwise, the Web community is doomed to experience all the delivery, quality and complexity problems that have plagued the software community.

The World Wide Web is a unifying force bringing the world closer together. Regardless of race, color, creed, skills, educational background, computer platform, browser, nature of business, geographical location, and job position, we all *look* the same. Business is being transformed into E-commerce. Such revenue is expected to hit $220 billion by 2001 [1]. Behind the facade of e-commerce though, the Web Crisis is looming [2]. It is the exponential proliferation of web content created, and maintained, without any expertise in data management techniques—the proliferation of *hacked together* web-based systems developed without any rigorous approach and kept running via a continual stream of patches. Companies are desperate to put their business applications on the Internet. *Gold rush fever* is encouraging business start-ups centralized around the Web. With the advent of many low-cost publishing tools that are very easy to use, web system creation is now so simple that anyone without programming skills can create one.

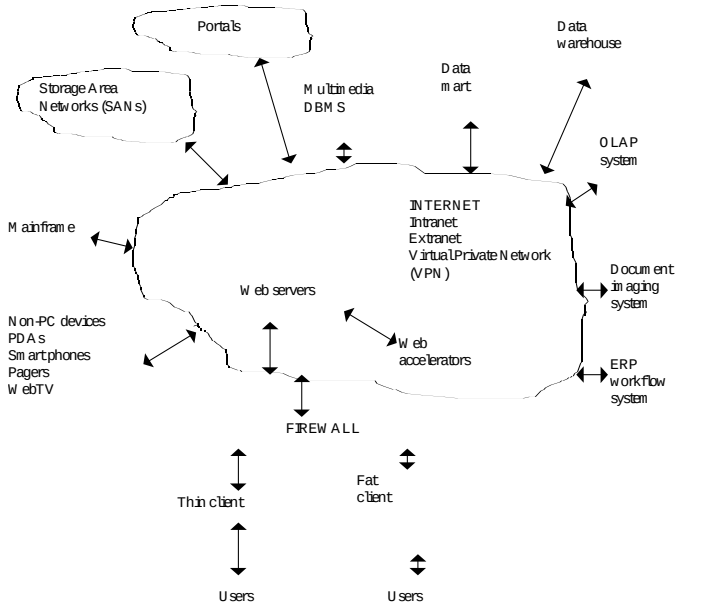
The demand for content creation and maintenance is escalating at an unmanageable rate. Some analysts have predicted that by the year 2002, the market revenue from content management tools will be around $5 billion [3]. And even when we have it under control, content has a multiplier or snowball effect, where we will further exploit new ways of using content. First-generation web systems have focused on providing access to any piece of information around the world. The next generation web systems will focus on knowledge management—managing the semantics, or concepts, of content rather than just the raw information.

For now, though, we see the shortcomings of first-generation web systems. There are problems with information being published on the web site at the wrong time and information that is inaccurate, top-secret, corrupt, inconsistent, unauthorized, unchecked, garbage, stale, or inappropriate. These can have devastating consequences for companies such as millions of dollars lost in lost revenue, lost customers, and lowered stock prices (such as with software crash [4]). The causes are easily linked to lack of well-defined processes, testing, cross-checking of information, authorized changes, security checking, or responsibility for coordinated changes. Essentially, the problems stem from poor CM practices. The first generation of web systems were crafted from immature tools and languages, and inexperienced staff. To properly provide Change Content Management (CCM)—CM for web systems—we will have to go beyond the capabilities traditionally provided by industrial-strength software CM tools because the challenges presented by the emerging web economy are exceptional.

This paper is designed to raise questions about CCM for the Web so that we can understand the new demands placed on companies by web systems. A web system is a generic term for an application that can be accessed via the Web. It fundamentally consists of content (its data, such as a document), application server (for executing actions on the data, such as updating document), access (its interface, such as the client's browser) and the web server (common ones include Apache, Internet Information Server and Enterprise Server). The biggest challenge for the Web community is how to build maintainable web systems that are highly responsive to immediate, high-volume change.

This paper defines the kinds of resources in the Web environment that are used in web systems, specifies the classes of web systems being developed, identifies the many challenges that companies are facing in their efforts to understand CCM, highlights capabilities provided by software CM and web CCM tools and ends with recommendations for approaching the solutions.

Figure 1. *The Web Environment*



The World Wide Web Environment

Web systems can be huge, with millions of pages, many interconnections, and incredibly high hit rates. Consider Figure 1 which highlights the many kinds of resources throughout the Web that can be components of web systems. It shows that users can be connected to the network via a thin client or a fat client. A thin client means application code is resident on the server, rather than on the client (fat client). A firewall determines the kind of access, encryption and security levels. Web servers provide much of the application code and can have accelerators for caching dynamic pages in order to improve user access time. The network can be specialized into an intranet, extranet or virtual private network (VPN). An Intranet is an internal network behind a firewall that allows only users within the company to access it. An Extranet allows outside partners to have access to the Intranet. A VPN is a secure and encrypted connection between two points across the Internet. It acts as an Intranet or Extranet except it uses the public Internet as the networking connection rather than a company's own wiring. This enables, for instance, a company's branch offices to be inexpensively connected via the Internet.

Attached to the network can be other types of networks such as storage area networks (SANs) and portals. SANs are networks that pool resources for centralized data storage. They may include multiple servers working against a centralized data store built with redundant hardware such as RAID (high-volume storage) devices. Portals (such as Yahoo!, AOL) are full-service hubs of e-commerce, mail, online communities, customized news, search engines and directories, all suited to the particular needs of an audience. Portals are evolving into corporate enterprise portals. Such portals, for instance, enhance corporate decision-making by integrating the company's applications, thereby removing barriers that exist between business units.

Other resources that can make up web systems are: Data Base Management Systems; workflow applications used for optimizing business processes, such as Enterprise Resource Planning tools (e.g., SAP, PeopleSoft, Baan); database applications such as OnLine Analytical Processing systems, which allow users to perform *multidimensional* analysis on data via their browsers; document management tools [5] for providing access into shared libraries of documents; imaging systems for optical character recognition of documents; data warehouses containing terabytes of data;[1] multimedia databases for holding archives of music, speech, videos; mainframes which contain approximately 70 percent of legacy data for large companies; data-marts, which are data warehouses with their own unique interpretation of business data to suit certain functional needs of a business unit; and, non-PC devices, such as pagers, personal digital assistants, WebTV, and smart phones.

Web systems are made up of various combinations of the resources shown in Figure 1. Each of the resources imply content that can be dynamically added, changed, deleted, accessed, manipulated, along with their relationships and hyperlinks. CCM will need to control the static content that goes into the web system along with the dynamic content that is created during execution of the web system. Different kinds of web systems are being developed which affect the nature of CCM.

Types of Web Systems

It is difficult to classify the types of web systems being built today as there is no universal blueprint for such systems, the design is still an immature art and the systems themselves are evolving fast. But for the purposes of opening up discussions about CCM, we need to understand the types of architecture of web systems with respect to content creation. In a broad sense, a web system which is visible via its web site, either acts as a provider of information or is an application. But the applications can be of different types.

From a content perspective, we are interested in types of web systems which have data points where data can be added, changed, deleted, accessed or accumulated. Once we understand the types of applications, we can then determine the nature of its CCM needs, its development processes, and types of tools needed to properly maintain it. A web system can be categorized as having the properties of one or more of the following classes:

- Informational: information sites with read-only usage, commonly called "brochureware" e.g., information presented on a site that gives details about a company and its products. First-generation web systems are this type and are static.
- Delivery system: download content to user or resource e.g., download upgrades or plug-ins
- Customized access: access is via a customized interface or based on user's preferences e.g., my customized view of my Internet Service Provider's home page, or favorite portal
- User-provided information: user provides content by filling in a form e.g., subscription to a magazine or registering for a company's seminar
- Interactive: Two-way interaction between sites, users and resources e.g., business-to-business
- File sharing: remote users collaborate on common files e.g., users coordinate schedules
- Transaction-oriented: user buys something e.g., buys books or travel tickets
- Service provider: rentable applications; user rents an application on a per user, per month basis e.g., virus scan program
- Database access: user makes queries into a database e.g., supplier looks up catalog of parts
- Document access: libraries of online documents are available e.g., view corporate standards
- Workflow-oriented: a process has to be followed e.g., order entry automation
- Automatic content generator: robots or agents automatically generate content e.g., "bots" scour the Web to bring back specific information such as best price on products.

Given these classes, it becomes obvious that content can essentially be created by anyone or any other resource: from the content designer, the webmaster, any user, another database or device, or other web system. From a CCM perspective, it is straightforward to capture content that makes up a released baseline since that is static content, but what about content that is created or changed dynamically? This raises four key questions:

(1)] What constitutes a configuration item for a baseline with static and dynamic objects?
(2) How can dynamic baselines be captured?
(3) Now that the user of the web system participates in the creation or changing of a baseline, how does that affect the definition of the CCM lifecycle?
(4) Are CCM requirements different for each class of web system?

These are some of the questions being asked by webmasters, developers and CM managers.

Enterprise Challenges for Web Systems

CCM is not a problem for small, static web systems managed by a few developers. It is a problem for medium and large, enterprise systems that involve many content developers creating many pages that will have a high hit rate involving high-volume database accesses and updates every minute. For instance, the NASDAQ stock exchange system [6], is a web system of types 1, 4, 5, 6,7, 9, 10 and 12, and was built to sustain 12 million hits per day with 8 web servers per database server. When the stock market goes *crazy*, the NASDAQ site gets 20 million hits per day. Its content must be completely accurate, and it changes within seconds. Boeing [7], with a web system of types 1, 2, 4, 5, 9, 10, and 11, has 1 million pages hosted by 2,300 Intranet sites on more than 1,000 web servers.

Developing and maintaining such large systems with large volumes of content offers many challenges to companies. These challenges span technical, people, process, and political issues. The major ones obvious today are the following, and are described in detail below.

1) the dynamic, active nature of content
2) variant explosion
3) the free-form style of development
4) the performance effect of content
5) scaleability of content
6) the urgency and frequency of change to content
7) the outsourcing and ownership of content
8) the immaturity of tools, techniques, standards and skills
9) corporate politics.

The Dynamic, Active Nature of Content

Web content is dynamic because it is created on-the-fly based on a user's or agent's request. It is active because programs are executed in response to the request and to the user's environment (browser and plug-ins on the client side). For instance, HTML is static but when combined with active controls (such as ActiveX), it becomes dynamic such as when the web site gives users feedback on the type of data they are supposed to enter to make sure the input complies. Content can be generated and changed in real-time such as with tables, forms, database queries, documents, and code.

Content is made up data objects, component libraries, and code. These can be static or dynamic, singular or a collection, compiled or interpreted, source or binary code. Objects include documents, images, streaming video and audio, files, or tables. Code can be active controls and scripts such as: ActiveX controls, Java, C++, VisualBasic, HTML, DHTML, XML, VRML, OLE controls, Active Server Pages, Java applets, VBScript, JavaScript, and ISAPI, CGI, and Perl scripts. Scripts or behaviours can be attached to web objects allowing, for instance, the user to alter attributes, such as color, positioning and font size on objects or execute applications. Component libraries are reusable code to be used as toolkits. Examples are JavaBeans, Microsoft Foundation Classes and Lotus' eSuite of business applets.

An applet or a control is a compiled binary file that a field in the HTML references. A script is executable code in a readable source language that can be embedded directly in the HTML tag. In essence, an object becomes a container for various pieces of content, all of which, need to be under CM control.

We are moving toward container-based, or a component bundling approach, to software development. This means CM techniques need to account for embedded scripts and customized components. Also, the executing environment needs to be taken into account. For example, if a browser does not support a certain scripting language, then the behavior of the web system will be different. Also, HTML files can be manually touched up. Scripts can easily be changed because they are interpreted whereas applets or controls typically are compiled. This assumes that the source code can be accessed, which is not the case when components are bought and reused in their binary form. Hence, changing or recompiling is not an option sometimes. Executing code may require a series of steps. For example, a Java file is compiled into platform-independent bytecodes; these are then processed by a Just In Time compiler to yield fast native instructions for a particular platform. All the above objects types, their relationships to intermediate forms, and all the tools, need to be tracked for good CM practices.

Web pages are dynamically created, which means that any CM control also must be of a dynamic nature. For example, an .asp file[2] is recognized; the VBScript is interpreted with the appropriate database or related files being accessed; the server creates the full HTML on-the-fly thereby dynamically generating the web page that is then displayed. How is all this data tracked for CM purposes? How is a dynamic baseline captured? To add complexity, hyperlinks can be created on-the-fly to point to documents. This changes the original baseline. Also, dynamically generated pages can be customized using ActiveX, CGI scripts, JavaScript and DHTML frames.

There is more. Content has meta-data associated with it that must be captured:

- The separation of content and format. Companies have stan dard templates into which content is published. These templates are part of the released baseline.
- External structure information, such as the hierarchy and relationship of web pages
- Internal structure information, such as embedded objects
- Hyperlinks to internal or external pages, static or dynamic
- Task objects that indicates some activity must happen to an object, such as updating the content
- Transaction, such as data involved in carrying out an e-commerce activity

- Security information attached to each objects
- Audit logs related to the activity on each object
- Tool compatibility information, such as the version of the browser for which this object is valid
- Bill of materials: the artifacts used to create the baseline (tools, tool options, data, files)
- Generated or converted files, such as a Word document that is converted into HTML
- Validation rules, such as a form requires input validation for each field
- Handler rules, such as a data base access request invokes certain tools and operations.

There are obviously many properties about content that need to be captured. Ideally, a company should have a well-defined CM data model that captures all the properties and relationships of content. With that, configuration items, baseline,s and releases can be defined.

Variant Explosion

Web systems imply a variant explosion problem. Consider that web systems are either created from scratch, are redesigned or merged web systems, or are web-enabled legacy applications. In many cases, companies must live with all these systems in parallel. Thus, a company could easily have a nightmarish number of versions of their latest baseline. For example, it has four variants of its main application available at all times:

- The demo version which is a partial web-enabled baseline of the original legacy code with a minimal set of functionality as this is the textual version
- a full version that is the same as the first where all functionality is available since it is the graphical version
- the true web version that is a completely redesigned form of the application ideally suited to the web rather thanmerely web-enabled legacy code, and
- the original legacy system for non-web use.

Each variant must work with two different browsers (Internet Explorer and Netscape Navigator), including the latest three versions of those browsers—and support five different languages for international use. Hence, we have (1 * 5) + (3 * 2 * 3 * 5) = 95 potential variants.[3] Most companies have different teams working on separate variants without much communication, reuse or change propagation across common code. With the variants, come all the complexity of parallel development support for simultaneous changes and concurrent baselines, along with significant change propagation to selected variants, thereby demanding change set support [8], more sophisticated change tracking along with help-desk support and much better release planning and change scheduling. The ramifications are dramatic. Variant management and change propagation have long plagued software companies.

The Free-Form Style of Development

Web system development is different from traditional software development. [9,10] This is due to the nature of the tools, languages, skills of the developers, and the dynamic nature of the Web environment. There is tremendous pressure on developers to code and publish. And the web tools support this free-form style of development. Also, the skill set of the developers is quite limited with typically no experience in software engineering. They are guided only by the capabilities of the tools and languages that, as we know from software engineering practices, cannot be adequate.

Scripting languages (such as JavaScript, Jscript, Tcl, VBScript) are changing the way that applications are developed. Most of these are interpretive languages or use Just In Time compilers. This leads to a style of change on the fly. There is no process between creating content and publishing it. Programming has gone from a process-oriented compiler-based approach, to combine components, mix in some new code and publish. Essentially, this squeezes the change cycle time dramatically because all sense of process is eliminated. This enables a faster rate of change that is a real benefit for web systems but provides greater opportunity for errors through lack of testing and content coordination and authorization of change. The question becomes how can testing, system integration, load testing and release management processes be inserted into the code-and-go paradigm to enable proper CM? Some companies use staging areas for testing before publishing to a live site, whereas many do not.

The complexity of web system development can be seen in Table 1. The major phases are highlighted along with who assumes responsibility for those steps. There are at least nine key steps involved in getting the web system functioning. At each point, CM issues come into play, such as, which release or version of the web system is being changed or published or tested or registered or validated for security purposes or being moni-

1. MAJOR ACTIVITY IN WEB SYSTEM	WHO DOES THE WORK
2. Design and creation	Web Team or IT Dept. or Outsourced
3. Infrastructure support: servers, network connections, databases	Outsourced to network management company, or hosted by IT Dept.
4. Testing e.g., compatibility of content, link accuracy, viewable by all kinds of browsers	Web Team or IT Dept.
5. Publishing of content	Business Units or Web Team or IT Dept.
6. Registering of sites on search engines	Web Team or IT Dept.
7. Security checking: access control, hacker analysis, virus detection	Web Team or IT Dept or Security Consultant
8. Monitoring: traffic performance: intelligent load balancing and web page redesign; replication; web accelerators/caching; traffic shaping capacity planning	Web Team or IT Dept.
9. Maintenance: content evolution via changes, enhancements, deletions, redesign	Content experts or Web Team or IT Dept.

Table 1. *Typical web system lifecycle phases*

tored for hits or performance improvements. Without CCM practices and tool support, these activities become fraught with errors. Automated workflow along with role-based activities must be supported in web tools

The Performance Effect on Content

Performance—particularly response time to a user's request—plays a major role in influencing content design. High performance web systems have continuous traffic monitoring. Users must have immediate access to quickly changing content under any load situations. If access times are not acceptable, a company makes a decision to either install web accelerators that enable caching to improve performance, or it redesigns the content for better access. For instance, at the Olympics site [11], traffic monitoring showed bottlenecks for users by having to navigate too many pages to get to the right content. The web site was redesigned on-the-fly to make access easier and speedier along with adding caches.

Web accelerators, or caches, are beginning to play bigger roles in performance enhancement, with content being designed to take into account caching techniques for accelerators. But a dependency results between the content baseline and the version of the caching algorithm and server that are used. Also, server crashes (such as with the E*trade brokerage site crashes that shut out users who lost money through lack of trading access) must be catered to in contingency plans. This means content must be replicated across servers that, in turn, means synchronization and distribution of real-time updates.

Scalability of Content

The Olympic and NASDAQ [12] web systems are huge in terms of number of pages, amount of traffic, and number of database and web servers. Millions of pages cannot be reasonably stored in a flat file system. Databases are obviously required for storage and are being redesigned to suit web access. Some database companies are redesigning their products so that web applications are stored directly in the database, such as Oracle's WebDB. This helps with scalability, reliability, and administration. It is likely that first-generation web systems will be redesigned to use web-enabled databases. This means that CM capabilities must be integrated and synchronized with database facilities.

The Urgency and Frequency of Change

The web enables the paradigm of change at the speed of thought. The mindset is typically: I see a problem and can, or need, to fix it immediately because it is globally visible. Corporate embarrassment or even worse, litigation, needs to be avoided. There may be no time to follow through a normal change life cycle (such as with a change request, Change Control Board, change authorization, edit, testing and re-release). Because the change can be done so easily, process is often bypassed. All the benefits of change tracking are lost. Repeatability will be a difficult benefit to achieve. Rollback of a site may be the only option for companies, but the corporate need of keeping the web site accurate takes top priority. There are changes that may need to be propagated across all pages of a web site, or just a few pages. For example, simply changing a copyright notice may involve changing each of the 1 million pages, whereas other changes may involve a select set of pages so that an incremental publishing capability is required, along with ways of organizing files into partitions to enable incremental updates. A company needs to define its classes and priority of changes and decide what process should be followed for each type of change.

Outsourcing

Outsourcing is a significant trend for industry, especially for web system creation, and sometimes maintenance. It is done for many reasons: to reduce operating costs, share risks with others, access leading-edge technology without having to purchase the infrastructure for it, use expertise not found in-house, do things more quickly, and to focus more on a company's core competencies. Outsourcing does require distributed management techniques along with doing CM with a third-party.

Easy-to-use web tools and specialized commercial-off-the-shelf tools (such as OLAP, ERP, document management) are helping to change the political infrastructure of companies. For instance, business units no longer are forced to rely on the Information Technology (IT) department in order to get things done. They buy the best tool that suits their need, bypassing IT. They can even rent the infrastructure for supporting the tools, and outsource its administration. This complicates issues of who has responsibility for what, how to maintain control and visibility over outsourced changes, and whether a business unit guarantees that a quality process was followed for the outsourced work.

Immaturity of Tools, Techniques, Standards, and Skills

Engineering techniques for web systems are in their infancy. Tools, standards, and skill sets are maturing, albeit slowly. Each month new tools and new versions of tools are released that support easier ways of building web systems. As a result, companies have to maintain different tool technologies in parallel. Standards (such as XML (eXtensible Markup Language) from World Wide Web Consortium, or WebDAV (from the Internet Engineering Task Force) are slowly being developed that in turn will affect the tools. There are many web technology tools that enable easy publishing of content without team coordination or process. Because of the many choices, large companies will end up having their business units using different tools. To get some control over how content is developed, and to ensure that quality processes are followed in publishing content, companies will have to define standards and guidelines. These standards will pertain to style templates, component libraries, tools, languages, servers, testing processes, and CM.

Web systems require developers and content experts for their creation and maintenance. Many web developers have little background in software engineering. Content creators can be human resources personnel, marketing people, accounting staff, etc. Their web skills are totally dependent on their knowledge gleaned from the web tool set and any training class they attended. This implies that the tools need to have interfaces that suit the content writer, yet have excellent CM processes embed-

ded to compensate for the lack of software skills.

Corporate Politics

There is confusion in companies these days as to who should have the right to publish content on the web site. For instance, business units publish independently from the IT department. Essentially there is lack of control as to what goes up, when, and how it has been tested and whether it conforms to standards. This is particularly a problem when the web system has content that must be coordinated and validated as a whole with other departments (accounting, marketing, personnel, etc.) or with other applications. Who assumes responsibility for the information's accuracy on the web site? Who assures that quality control processes have been followed before information is published on the site? Who is responsible for making changes? Who assumes the cost of change? The IT department's role is changing dramatically—from an infrastructure provider to that of a strategic advisor and standards producer. Many traditional IT functions, such as network administration, are being outsourced. Outsourcing will significantly change the modus operandi of IT departments. Web creation is mostly outsourced these days. Companies face a delicate balancing act in trying to rein in the proliferation of web systems while leaving employees freedom to meet their business needs.

Software CM—Major Part of Content Change Management

Everything that the software community has learned about CM can be applied to the CCM problems. Software CM spans a significant spectrum of activities and roles within a company [13, 14] and Table 2 highlights the main goals of CM. The software CM tool vendors are adding CCM capabilities to their tools.

GOAL	EXPLANATION
Identification	Uniquely identify parts of the content
Control	Version control of all objects including baselines
Status accounting	Tracking the status of all work on all objects
Audit and review	Keeping an audit trail, confirming all processes followed
Cost-effective production	Fast and quick builds of software releases
Quality automation	Ensuring all testing, notifications, signoffs, reviews are done
Teamwork optimization	Enabling teams to work in parallel effectively
Enabling change	Containing the explosion of changes

Table 2. *Goals of software configuration management*

Web tool vendors are beginning to realize that CM practices must be incorporated into their tools Advice on good web design [15] is beginning to highlight the importance of CM but only in the sense of version control of files. However, according to Powell, web engineering advice completely ignores CM. Table 3 lists some of commercial software CM tools.

CM TOOL	VENDOR	WEBSITE
Continuus	Continuus	www.continuus.com
ClearCase	Rational	www.rational.com
Harvest	Platinum Technology	www.platinum.com
Perforce	Perforce Software	www.perforce.com
PVCS	Merant	www.merant.com
Source Integrity	MKS	www.mks.com
SourceSafe	Microsoft	www.microsoft.com
StarTeam	Starbase Corp.	www.starbase.com
Team Connection	IBM	www.ibm.com
TrueChange	True Software	www.truesoft.com

Table 3. *Some Commercial Configuration Management Tools*

Software CM vendors are taking different approaches to CCM support in their tools. Some, such as StarTeam, are web-enabled and have purchased web technology companies with the intention of tool integration. Others, such as TrueChange, have decided to build a completely new software CM tool for CCM. Others such as Continuus and MKSIntegrity, have added on CCM support. The former's offering, WebSynergy and WebPT, provide a web front-end into all of its existing CM process-oriented capabilities as well as web-authoring tools with transparent access to files. The latter's offering, WebIntegrity, integrates its version control facilities with an authoring tool.

Web Technology Tools

Web tools are marketed for web authors or web developers. As to what constitutes a CCM tool, that is not totally clear and there is no consistency in functionality across the tools. Suitability for large-scale development seems to determine whether it is a CCM tool or an authoring tool. Tools are first generation ones (with respect to CCM support), with only one product (DynaBase) claiming that it provides configuration management facilities. Some tools are geared to large-scale web production although it is not yet clear how scaleable these tools are. Half a million components seems to be the maximum now. Table 4 lists some commercial CCM tools.

If there are any similarities or trends, they would be:

- support for web languages
- command line interfaces
- templates for separating content from formatting
- version control of files
- roll-back of complete sites
- minimal workflow support for publishing authorization
- audit logging, event triggers
- commercial database interfacing
- drag-and-drop component reuse (to minimize programming)
- role support for authorizations
- minimal change tracking
- concurrent site production (for multiple releases).

Some noteworthy features include:

- TeamSite provides visual differencing for examining two versions of content side by side.
- Tasks can be assigned to authors using notifications.
- Authors can be notified when content is published on the web.
- Content is moved to a staging area each time it is changed or receives approval to be published.
- Drumbeat gives developers guidance on targeting code to specific browsers thereby providing variant creation support.
- Raveler teams can be set up with pre-configured workflows.
- StoryServer supports static and dynamic versioning.

Overall, more CM support needs to be provided to support CCM needs.

Conclusion

The web environment provides the opportunity to connect

many different resources. Whilst the resultant web systems are easily created, they are complex systems offering many challenges for CCM. We need to understand the problems that companies are having with web systems in order to properly define their CCM requirements. We still need solutions to questions such as:

1) What are good content development and change processes for teams developing large-scale web systems?
2) Are there different processes depending on the type of web system, size of company, volume of web data?
3) Can the types of web systems be categorized into classes or architectures?
4) Will component libraries be indicative of these architectures?

CONTENT TOOL	VENDOR	WEB SITE
ArticleBase	Running Start	www.runningstart.com
DreamWeaver	Macromedia	www.dreamweaver.com
Drumbeat 2000	Elemental Software	www.drumbeat.com
DynaBase	Inso	www.inso.com
Frontier	Userland	www.userland.com
FrontPage98	Microsoft	www.microsoft.com
Fusion	NetObjects	www.netobjects.com
Raveler	Platinum Technologies	www.raveler.com
StoryServer	Vignette corp.	www.vignette.com
Team Site	Interwoven	www.interwoven.com

Table 4. *Commercial Web Content Development Tools*

5) What factors affect the definition of the CM process and CM items?
6) Do we need system models, data models, architectures of web sites in order to fully capture the appropriate CM meta-information?

Second-generation web systems will focus on knowledge management and need sound engineering principles such as CM behind them. Given the many challenges, much of the solution will have to be embedded in the tools because the skill set of the developer cannot be guaranteed. This means that the CM processes will have to be implemented in the web tools rather than relying on manual procedures. Along with excellent variant support, change tracking, and change propagation (especially via change sets). CM is becoming an issue for all companies because in order to survive beyond the first decade of the new millenium, companies must place their applications on the World Wide Web.

References

1. International Data Corporation, 1999.
2. Murugasen, S., Deshpande, Y.: *Proceedings of ICSE99 Workshop on Web Engineering.* International Conference on Software Engineering, Los Angeles, USA (May 1999)
3. Merrill Lynch Co., 1999.
4. Bloomberg News: Net Shares Battered Amid Signals That Web's Expansion Is Slowing. *Wall Street Journal* (June 15, 1999)
5. Dart, S: The Dawn of Document Management. *Application Development Trends* (Aug. 1997)
6. Hutcheson, M.: The NT Application That Wouldn't Die (NASDAQ.COM). *Enterprise Development.* 1,1 (Dec. 1998)
7. Sliwa, C: Maverick Intranets: A Challenge for IT. *Computerworld* (March 15, 1999)
8. Dart, S.: To Change Or Not To Change. *Application Development Trends* (June 1997)
9. Gellerson, H. Gaedke, M.: Object-oriented Web Application Development. *IEEE Internet Computing* (Jan/Feb 1999) 60-68
10. Lockwood, L.: Taming Web Development. *Software Development Magazine* (April 1999)
11. Iyengar et al.: Techniques for Designing High-Performance Web Sites. *IBM Research* (March 1999) 17pp
12. Powell, T.: *Web Site Engineering.* Prentice Hall, NJ, (1998)
13. Dart, S.: The Agony and Ecstasy of CM. A half-day tutorial given at 8th International Workshop on Software CM, Brussels Belgium (July 20-21, 1998) http://www.cs.colorado.edu/~andre/SCM8/dart.html
14. Dart, S.: Not All Tools are Created Equal. *Application Development Trends* (Oct. 1996) 7pp http://www.adtmag.com/pub/oct96/fe1002.htm
15. Siegel, D: *Secrets of Successful Web Sites : Project Management on the World Wide Web.* Haydn Books, Indianapolis, Ind. (1997)

Notes

1. Data warehouses provide common interfaces to variant databases.
2. Active Server Page, a combination of static HTML and VBScript
3. Then add in variants for non-PC devices such as pagers, PDAs and smart phones that have "micro-browsers", and the number of variants escalates further.

About the Author

Susan Dart is President of Dart Technology Strategies Inc., an independent consulting firm that helps companies attain configuration management solutions. Dart has 23 years of experience with software tools, development environments, and technology adoption. She has several publications and seminars to her credit, including *Evaluating Configuration Management Tools* (Ovum, 1996), and was a member of the U.S. Federal Aviation Authority committee on developing CM for bomb detection systems. She is a member of the Program Committees for the International Conferences on Web Engineering and on CM.

Previously, Ms Dart was Vice President of Process Technology at Continuus Software Corporation, where she implemented deployment services to assist strategic customers in achieving the best possible, enterprise-wide CM solution. Before that, she spent seven years at the CMU SEI, creating models for CM and evaluating software development environments. Dart has also developed compilers at Tartan Inc. and telecommunications software and international standards for Telstra, Australia. She has a master's degree in software engineering from CMU and a bachelor's degree with distinction in domputer science from RMIT.

Dart Technology Strategies Incorporated.
1280 Bison, PMB 510.
Newport Beach, Calif. 92660. USA
Voice: 949-224-9929
Fax: 949-515-4442
E-mail: sdart@susandart.com
Internet: www.susandart.com

The Need for a Useful Lessons Learned Database

George Jackelen
EWA Inc.

Literature and standards mention the importance of lessons learned. So why are we still having problems? Why have we not learned our lesson?

Scene: "The project is finally over," the project manager muses. "Everyone has either been transferred, is hanging around to wait for the next assignment, or is satisfying the old saying, 'The job ain't done until the paperwork is finished.'

"Hmm. That just leaves preparing the final costing reports, archiving the products and related paperwork in case there is a post-audit, and preparing lessons learned. Lessons learned should only require a page or two; I'll find out who is available."

An Afterthought

As shown by the above example, documenting lessons learned is often downplayed as a needed, and sometimes required, task to help future projects avoid problems. The need for lessons learned is described as:

> "… so that they [causes of variances, the reasoning behind the corrective action chosen, and other types of lessons learned] become part of the historical database for both this project and other projects of performing organizations"[1].

So, what is the problem? My experience in the federal government and industry is that the current lessons-learned efforts are not seen as worth the time spent in their implementation. Lessons learned are mainly afterthoughts done at the end of a project to close out one of the final checklist items to terminate a project. Providing lessons learned when they occur is considered to be time consuming, and there are things of higher priority that must be completed first. There is a prevailing view that the idea of having lessons learned is good. In reality, presented issues are normally typical (for example, schedule and cost must be managed) or unique to a project.

As has been stated by many authors and project managers, "It is generally cheaper to prevent problems than to fix problems." This is especially true of project problems. If done correctly, an organizational lessons learned database can prevent project headaches and save time and money.

Current Lessons Learned Task

What causes these obstacles? Let us take one example, "What is wrong with looking at old problem reports?" Problem reports do not provide lessons learned information since management and business issues are normally not part of problem reports. Also, many problem report issues are so technical and project specific that they do not help future projects anticipate or avoid problems. As a result, many problem-report solutions are also too technical and project specific. In addition, most problem reports do not address the question, "What could have been done to prevent this problem?"

Proposed Lessons Learned Task

To overcome these obstacles and to make a lessons learned task work, there must be a clear understanding of the purpose for documenting lessons learned. I propose the following definition:

> Lessons learned task—An ongoing project task to document a project's major negative and positive issues. The documented lessons learned are used to prevent issues from having a negative impact on a project and to provide alternative ways of doing things.

To implement a continuous lessons learned task, without making it an overhead nightmare, it is necessary to define the type of issues to be continuously reported. For example, issues approaching a plus or minus 5 percent impact on cost, schedule, or deliverable product size or performance shall be entered into an organizational lessons learned database.

A second lessons learned subtask is to consider the format of a lessons learned report. A problem report identifies a problem, the cause, and solution; a lessons learned report identifies the same things as well as what should have been done in advance to recognize and prevent the issue.

A third lessons learned subtask is that a successful lessons learned database needs to have the assignment of responsibilities. Someone in a project management office could be responsible for cost, schedule, customer interactions, and management lessons learned. A quality assurance organization could be responsible for deliverable product issues (including processes, development, delivery, installation and maintenance).

Thus, a lessons learned task does not have to be complicated, costly, or wait until project termination.

Is It Worth the Effort?

A timely lessons learned database must be part of an organization's process improvement effort. This database can be used to reduce risks by providing users with information about how people were able to recognize risks (hopefully in advance) and overcome these risks. As a result, a lessons learned database becomes not just a project tool, but also an organizational tool to help ensure the past can be used to help the future of an organization. An organizational lessons learned database can economically reduce risks and costs while providing greater benefits than independent project lessons learned databases.

Recommendation

A common statement is that quality must be built into a product or service. For product quality to occur, a person/ organization must step back and realize that quality must be built into a

project before quality can be built into a product or service. Going a step further, quality must be built into an organization before quality can be built into a product. The following steps can help establish quality projects:

- Define the format and criteria for information to be stored in an organizational lessons learned database.
- Assign responsibility for timely updates of the lessons learned database to a few people or organizations. The responsibility should include examining an organizational lessons learned database prior to and during a project.
- Identify metrics, including frequency of measurement and reporting, for ensuring effective lessons learned task implementation and to measure the level of success. For example, have project metrics to identify the lessons learned database's usefulness. This could include such questions as:
  - —What issues were prevented or resolved, based on the provided information?
  - —What information is being duplicated? Does the organization have a bigger problem?
  - —Is the lessons learned database periodically audited to ensure it is being updated with timely information that is used?

Conclusion

A lessons learned database is a useful management tool to help projects reduce the number and level of risk items and to provide useful information to identify positive ways to run a project. The main need is for a positive attitude about the value and cost effectiveness of efficiently run lessons learned databases. In addition, there can be a tremendous organizational benefit to reduce risk and to assist in a process improvement effort. There is a need for useful lessons learned databases.◆

Acknowledgement

I want to thank Dan Solomon for his advice.

Reference

Project Management Institute. *A Guide to the Project Management Body of Knowledge*, Project Management Institute, Upper Darby, Pa., 1996.

About the Author

George Jackelen is project manager and analyst for two NASA Independent Verification and Validation projects. During his 30-plus years experience, he has performed software and hardware quality assurance for DoD and industry, and developed or provided review comments on ISO, IEEE, DoD and contractor standards, policies, procedures, etc. He has a master's of science degree in Computer Science from Texas A&M University.

1000 Technology Drive
Fairmont, W.Va. 26554
Voice: 304-367-8252
Fax: 304-367-0775
E-mail: gjackele@ewa.com

Coming Events 2000

February 28–March 3
16th International Conference on Data Engineering ICDE 2000
http://www.research.microsoft.com/icde2000/

March 6-8
13th Conference on Software Engineering Education and Training CSEET&T
http://www.se.cs.ttu.edu/CSEET2000

March 6-10
Software Management/Applications of Software Measurement SM/ASM 2000
http://www.sqe.com/smasm/2000/

March 13-17
6th International Conference on Practical Software Techniques
http://www.softdim.com

March 20-22
5th Annual Association for Configuration & Data Management Technical Conference
http://www.acdm.org

March 20-23
12th Software Engineering Process Group Conference
SEPG 2K
http://www.sei.cmu.edu/products/events/sepg/

April 11-14
Infosecurity 2000
http://www.infosec.co.uk/page.cfm

April 18-20
FOSE :Leading-Edge Technology for Leaders in Government
http://www.fedimaging.com/conferences

April 30–May 4
The 12th Annual Software Technology Conference
http://www.stsc.hill.af.mil/index.asp

June 20-22
Product Focused Software Process Improvement PROFES 2000
http://www.ele.vtt.fi/profes2000

August 28-31
First Software Product Line Conference SPLC1
http://www.sei.cmu.edu/plp/conf/SPLC.html

CrossTALK

SPONSOR Lt. Col. Joe Jarzombek

PUBLISHER Reuel S. Alder

ASSOCIATE PUBLISHER Lynn Silver

MANAGING EDITOR Kathy Gurchiek

ASSOCIATE EDITOR/LAYOUT Matthew Welker

ASSOCIATE EDITOR/FEATURES Heather Winward

VOICE 801-775-5555 DSN 775-5555
FAX 801-777-8069 DSN 777-8069
E-MAIL crosstalk.staff@hill.af.mil
STSC ONLINE http://www.stsc.hill.af.mil
CROSSTALK ONLINE http://www.stsc.hill.af.mil/Crosstalk/crostalk.html
CRSIP ONLINE http://www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address:
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, Utah 84056-5205
E-mail: stsc.custserv@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*.

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: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

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 Online Services: This 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.

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available 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, Utah 84056. CROSSTALK is distributed without charge to individuals actively involved in the defense software development process.

Software Pie

With appreciation and apologies to Don McLean . . .

A long, long time ago
I can still remember how that PC used to make me smile
And I knew if we got a chance
We could make those pixels dance
And maybe we'd be happy for a while
But February made me shiver
With every program he'd deliver
Bad news on the Altair
It just didn't seem fair
I can't remember if I sighed
When I read about his legal slide
But something touched me deep inside
The day the software died
So … {Refrain}

Refrain

Bye-bye, Mr. Microsoft Guy
Got his product preloaded
And I still don't know why
And them Redmond boys were eating ham on rye
Singin' Bill is just a regular guy
He even stuck his face in a pie

Did he write BASIC stuff
And did he have faith in DOS enough
If Allen told him so
Do you believe in point 'n click
Can GUI save your carpal tick
And can you teach me how to surf real slow
Well, I know that you're in love with DOS
'Cause I saw you sell it to the boss
He had no clue to choose
Man, I dig those IBM blues
He was a lonely teenage techno-geek
With a gift for fancy market speak
But I knew it was tongue-in-cheek
The day the software died
I started singing … {Refrain}
Now for 10 years we've been on PCs
Still DOS grows fat on our CDs
But that's not how it used to be
When Gates brawled with the Apple King
In a coat worn since he was 13
And cash that came from you and me
Oh, and while the King was looking down
Gates stole his thorny crown
The courtroom was adjourned
No software was returned
And while Netscape geared up on the Net
The Redmond boys began to sweat
Their next move would set their debt
The day the software died
We were singing … {Refrain}
I met a girl who runs Outlook
And I asked her to fix my address book
But she just smiled and turned away
I went down to the sacred store
Where I'd run Outlook once before
But the man there said the program wouldn't run
And in the streets the users screamed
The managers cried, and the hackers dreamed
The servers were all chokin'
The firewall had been broken
And the two men who had the most,
Bill the kid and Paul the ghost
They built their mansions on the coast
The day the software died
And they were singing …
Bye-bye, Mr. Microsoft Guy
Got his product preloaded
And I still don't know why
And them Redmond boys and their long hippie hair
Singin' this will be the day that we cry

—Gary Petersen, Shim Enterprises

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

Policy and Management

Lessons Learned

Software Engineering Technology

Web Addition

Open Forum

Departments and Announcements

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

BULK RATE
US POSTAGE PAID
Permit No. 481
Cedarburg, WI