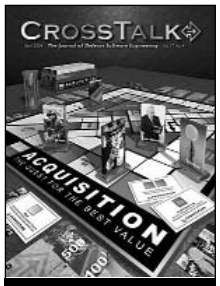
CROSSTALK

April 2004 The Journal of Defense Software Engineering Vol. 17 No. 4

Acquisition



ON THE COVER

Cover Design by Kent Bingham.

Software Engineering Technology

Open Forum

Departments

CROSSTALK

PUBLISHER	*Tracy Stauder*
ASSOCIATE PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Pamela Palmer*
ASSOCIATE EDITOR	*Chelene Fortier-Lozancich*
ARTICLE COORDINATOR	*Nicole Kentta*
CREATIVE SERVICES COORDINATOR	*Janna Kay Jensen*
PHONE	(801) 586-0095
FAX	(801) 777-8069
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/crosstalk

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 15.

Ogden ALC/MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

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 Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does 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: www.stsc.hill.af.mil
Call (801) 777-7026, e-mail: randy.schreifels@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available free of charge.

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 to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Contract Oversight Requires Data

I am concerned when I learn about acquisition programs that do not have adequate insights into development. In a recent discussion, I learned that a government group acquiring software was developing measurements they would use for their own quality assurance group. When I pointed out that their measurements were deficient because none of them tied directly to the software being developed, I was told the acquisition organization had no authority to obtain insights to the quality measurements from the developers.

As an acquisition organization, it is your right and *responsibility* to be certain you have all the data necessary to ensure the program is proceeding as it should. The organization mentioned above was correct in some respects because the developers are not required to provide this information if it is not stated as a deliverable in the contract (and apparently it was not stated in this contract). I would like to send a message now that all new contracts are negligent if they do not require the contractor to provide the data needed to oversee the contract. While current rules no longer allow acquisition organizations to stipulate data format, you can still require information related to cost, schedule, quality, risk management, etc. If the developers consider this to be proprietary information, you can sign a nondisclosure agreement, but you should require the information.

This also should not result in substantial additional costs to the program. I know of one program where the acquisition organization simply required a copy of the developer's software development plan and access to their databases. The acquisition organization was able to use the development plan to learn what data was available and where it was located. They then looked for the information as they needed it. The result was minimal additional cost.

Our first article this month is a discussion on Section 804 of the Bob Stump National Defense Authorization Act for Fiscal Year 2003. Lisa Pracchia's *Improving the DoD Software Acquisition Processes* discusses some of the issues that resulted in 804 being passed, the intent of 804, and some suggestions for implementing it.

Our next article is *Why We Need Empirical Information on Best Practices* from by Dr. Richard Turner. This author's experience in the Office of the Under Secretary of Defense has given him many opportunities for insight into programs going right or going wrong. In this article, Turner suggests some questions we should ask ourselves before jumping onto the latest *best practice*.

In *A Project Risk Metric,* Robert W. Ferguson reminds us of the need for managing risks throughout a project and suggests some values to quantify the actual risks.

In our supporting articles, John S. Willison discusses some benefits his project has seen while implementing agile development in *Agile Software Development for an Agile Force*. Michael S. Russell then provides criteria to use while considering reuse software in *Applying Decision Analysis to Component Reuse Assessment*. Michael J. Hillelsohn provides an additional benefit to good requirements management in *Better Communication Through Better Requirements*. Requirements management is essential to any software development. However, have you considered that better requirements' reviews of requirements could also aid understanding between the acquisition and development organizations? Finally, D.B. Robi reminds us that we need a reason for many of the best practices that we are asked to implement. He states a motivational view should be added to the Department of Defense (DoD) Architecture Framework in *Enterprise DoD Architecture Framework and the Motivational View*.

With Section 804 of the National Defense Authorization Act, there is more pressure on acquisition organizations to take responsibility for taxpayer dollars. I hope they will take this responsibility seriously and start getting the information they need to oversee a successful project. Four key points to remember are 1) know the rules for acquisition; the Federal Acquisition Regulation and Defense Federal Acquisition Regulation Supplement are too long to read and memorize, but an overview and retention of key points applicable to your project are essential, 2) tailor the data item descriptions for your needs; do not let the weight of the whole bureaucracy overwhelm the project, 3) work closely with your Acquisition Center of Excellence and your legal office's Contract Law Division, and 4) stay current on new acquisition approaches and review lessons learned during development. CROSSTALK's Web sites on Page 8 provide sources to help with these key areas.

Elizabeth Starrett

Elizabeth Starrett
Associate Publisher

Improving the DoD Software Acquisition Processes

Lisa Pracchia
Naval Air Systems Command

While the U.S. Department of Defense (DoD) weapons are undeniably superior, programs to acquire them continue to experience cost overruns, schedule slippages, and performance difficulties[1]. Improving software acquisition processes to address these issues was mandated in Section 804 of the National Defense Authorization Act for Fiscal Year 2003 and enacted on Dec. 2, 2002. This article explains the history leading to that public law, provides insight into Congressional intent, and outlines DoD guidance for implementing Section 804.

Recent military operations around the world demonstrate the superiority of U.S. weapon systems developed by the Department of Defense (DoD). Furthermore, an ever-increasing percentage of the weapon systems' functionality is provided by software, which constantly becomes more sophisticated and complex. While the DoD has risen to the challenge, cost overruns and unsatisfactory performance have led the General Accounting Office (GAO) to designate the DoD systems development and modernization efforts a high-risk area [1].

Significant risk factors include the enormous size and complexity of the software within these systems and acquirers' inadequate, inefficient, or unexpected processes for managing software-intensive system acquisitions. As one congressional source said when describing the acquisition of U.S. weapon systems, "It's not about bending metal any more, it's about routing electrons."

Software enables a myriad of complex capabilities from massive data fusion across geographically disparate large-scale sensor systems, to decisional systems that automatically select the most appropriate weapon and platform to attack a given target, to autonomous systems that operate without human intervention to destroy incoming missiles. Software creates the network-centric operation – the cornerstone of the DoD's transformation.

Several root causes for the GAO's designation point to long-standing *cultural* issues (culture being defined as the collective patterns of behavior exhibited by the numerous participants in the acquisition process and the incentives for their behavior). These cultural issues were highlighted in 1992 GAO reports [2, 3]. Two of these still-relevant issues are the acquisition community's bias toward hardware, and the fact that the community addresses critical software issues too late in the acquisition process.

In a 1998 CROSSTALK article [4], Capers Jones defined a major DoD system as having 12.5 million C Statements[2] (roughly the size of a major computer operating system of that day) and a development team that numbered in the hundreds. Typically, lack of process and intergroup communications was a problem; paperwork and software rework absorbed the bulk of development costs. Formal configuration control and change management were expensive and poorly implemented for projects that large. The probability of termination for one of those major software-intensive systems, Jones said, was 65 percent; he cited poor project management and inadequate quality control as primary factors.

Fast-forward five years to today's jointly developed system of systems. Take, for example, the Army's Future Combat Systems (FCS), a joint Army/Defense Advanced Research Projects Agency program. The Army's vision for the FCS is to create an integrated battlespace where networked information and communications systems provide a competitive edge to soldiers in the field and commanders in the control room. You would be hard pressed to even try to estimate the numbers of FCS developers as its extended team consists of one prime contractor, eight major subcontractors, and 55 other companies under contract [5].

According to congressional sources, "The FCS is estimated at 32 million total SLOC," or software source lines of code. The actual number, however, will likely be greater as past experience with software estimation has shown that we typically both underestimate size and add functionality as the development progresses.

Successful fielding of the FCS requires more mature acquisition, development, and testing approaches than used in the past for smaller systems. Previous approaches simply will not be adequate to guarantee that development cost, schedule, and performance baselines are met. Specifically, greater effort will have to be spent on managing changes to requirements and ensuring that information is shared among all stakeholders. What does all this mean to both the program offices and Congress? Mature processes must be used to ensure that the system functions as intended, and that major problems and errors are caught well in advance of operational tests.

Given that software-intensive projects are among the most expensive and risky undertakings of the 21st century, the investment in weapons from fiscal years 2003 through 2009 will exceed $1 trillion [6]. Furthermore, many of the DoD's most important technology projects will continue to deliver less than promised unless changes are made [7]. Improving how we acquire software-intensive systems is both long overdue and an imperative.

The History

Software Development Process Improvement

In the late 1980s, software developers began investing in process improvement by adopting best practices. Many public and private organizations based their improvement programs on the Software Engineering Institute's (SEI[SM]) Capability Maturity Model® (CMM®) for Software (SW-CMM). While adoption was slow at first, by the mid-90s companies with improvement programs were showing results.

For example, SEI reported in 1995 [8] that a major defense contractor that implemented a process improvement program in 1988 had reduced its rework costs from about 40 percent to about 10 percent of total project cost, increased staff productivity by 170 percent, and reduced defects by about 75 percent over a seven-year peri-

[SM] SEI is a service mark of Carnegie Mellon University.
® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

od. According to a 1999 SEI report [9], a software development contractor reduced its average estimated schedule deviation from 112 percent to 5 percent between 1988 and 1996. During that same period, SEI reported that this same contractor reduced its average estimated cost deviation from 87 percent to minus 4 percent.

By 2001, software development units within the DoD were also showing results from their improvement programs. According to one GAO report [10], each DoD unit with a software process improvement (SPI) program reported positive effects on software/systems quality. The Defense Finance and Accounting Service, for example, reported that its SPI program had reduced its overall software delivery cost by about one-third less than organizations of similar size; one Navy software activity reported reduced costs, improved product quality, and a 7:1 return on its SPI investment; and an Army activity reported that it had almost doubled its productivity in writing software for new systems because of improvements made under its SPI program.

Software Acquisition Process Improvement

While many defense and civilian contractors *developing* software-intensive systems have made performance gains through SPI, those *acquiring* these same systems have lagged behind. In situations where acquirers with a low level of process maturity contract for software from developers with a high-level process maturity, problems occur. For example, acquirers may try to circumvent development and management processes because they feel that following the process impacts their ability to meet the goal. The result of this *process avoidance* by the acquirer can be rework, additional delays, or unexecutable cost and schedule quotes – exactly what the process was designed to avoid had it been followed.

Other problems can occur at the end of the development process. If cost and delivery schedules become more important to the acquirer than having the developer meet their exit criteria for delivering a quality product, then the result can be software delivered with avoidable defects. An acquirer with a low process maturity is at a greater risk of having its program meet schedules and costs, but fail to deliver required performance.

The GAO has been reviewing weapon systems investments for more than 20 years. What they have found are consistent problems – cost increases, schedule delays, and performance shortfalls – along with underlying causes such as pressure on program managers to promise more than they can deliver [6]. In recent years, several of those reports have included consistent recommendations to implement best practices for software-intensive systems acquisition, and to initiate broad improvement programs.

In a 2001 report to the Armed Services Committee, for example, the GAO recommended that DoD establish and implement a department-wide SPI program based on accepted best practices [10]. In response to GAO's recommendations, the DoD identified two existing groups within the Office of the Secretary of Defense (OSD)[3, 4] as appropriate places for SPI to be addressed. The DoD also pointed to a revision of DoD Regulation 5000.2-R (since cancelled) as the needed policy guidance for improving software. The author believes that subsequent DoD inaction in response to GAO's recommendations played a pivotal role in Congress legislating software acquisition process improvement.

"Given that software-intensive projects are among the most expensive and risky undertakings of the 21st century, the investment in weapons from fiscal years 2003 through 2009 will exceed $1 trillion."

On Dec. 2, 2002, Section 804 of the National Defense Authorization Act for Fiscal Year 2003 [11] (or simply *Section 804*) was enacted. The Senate report accompanying its version of the National Defense Authorization Act for Fiscal Year 2003 [12] was clear on its intent and purpose. The report articulated the Senate's concerns with the negative impact of longstanding software problems on major defense acquisition programs. The Senate noted the recommendations from [10] and stated that the purpose of Section 804 was to implement the GAO's recommendations.

Section 804: The Law

Section 804 mandates improvement of the DoD's software acquisition processes. This legislation directly instructs the secretaries of each military department and heads of selected defense agencies to establish software acquisition process improvement programs – an apparent message of frustration with the way software improvement had been handled in the past.

Software acquisition process improvement program requirements include the following:

- A documented process for software acquisition planning, requirements development and management, project management and oversight, and risk management.
- Efforts to develop appropriate metrics for performance measurement and continual process improvement.
- A process to ensure that key program personnel have an appropriate level of experience or training in software acquisition.
- A process to ensure that each military department and select defense agency implement and adhere to established processes and requirements relating to the software acquisition.

Section 804 also requires that the assistant secretary of defense for Command, Control, Communications, and Intelligence, in consultation with the undersecretary of defense for Acquisition, Technology, and Logistics do the following:

- Provide applicable improvement program administration and compliance guidance, and ensure that secretaries of the departments and selected agencies comply with that guidance.
- Assist the departments and agencies with their respective improvement programs by ensuring they use applicable source-selection criteria and have access to a clearinghouse for information regarding best practices in software development and acquisition in both the public and private sectors.

Congressional Intent

Norm Brown, founding director of the former Software Program Managers Network, and Navy department member of the 2000 Defense Science Board Task Force on Defense Software said:

> Anyone looking at the past congressional actions and listening to the frustration expressed in congressional hearings will find the fundamental improvements mandated in Section 804 come as no surprise. The only surprise is that Congress has been as patient as they have been. Now, congressional patience seems to be turning to impatience;

> an impatience to see significant improvement in fixing our perennial problems with cost, schedule, and performance – and in addressing the underlying drivers that are causing these problems.

Congressional sources affirm that:

> ... [the] DoD is going to have to pay attention from the ground up, in other words, at the program manager level, or programs will continue to get tanked. Congress will remain interested and we're not going to let this go until [the] DoD significantly improves how it acquires software-intensive systems. The only way it's going to get fixed is by people on the inside – it simply makes no sense on any level to continue to ignore it.

Another indication of Congressional intent is the GAO's tasking to monitor the DoD's compliance with Section 804. Initially, the GAO was tasked to evaluate the DoD's efforts to develop programs for improving software acquisition processes and to assess how those efforts compared with leading commercial companies' practices. This initial GAO report (GAO-04-393) was scheduled for publication in March 2004. Subsequent GAO assessments will likely focus on compliance with specific Section 804 requirements.

Implementation

DoD Guidance

On March 21, 2003, the DoD issued a memorandum to provide the uniform implementation guidance that Section 804 requires. This memorandum identified applicability, delineated organizational roles and responsibilities for overseeing implementation, and clarified initial expectations for improvement programs. It also instructed military departments and selected defense agencies to establish software acquisition process improvement programs. Requirements for these programs included defining and applying measures, following applicable methods based on some structured approach that included an appraisal method, and determining and reporting status of process adherence and performance effectiveness.

The DoD memorandum also gave the OSD Software Intensive Systems Steering Group the role of leading a DoD-wide effort to improve software acquisition processes. This role entailed providing program guidance; identified best practices; established a clearinghouse of information regarding best practices and lessons learned in software development and acquisition; and provided guidance for documenting, performing, and continuously improving a minimum of eight specific software acquisition processes (the original four processes called out in Section 804, plus four additional processes[5]).

General Approaches

The OSD's implementation guidance has not been prescriptive. Component and agency approaches to compliance vary widely. That variety is clearly illustrated by the list of best practice models selected as the basis for software acquisition improvement programs. Model selections range from the IDEAL[SM] Model[6], to the CMM Integration[SM] (CMMI®)[7], to the Software Acquisition CMM (SA-CMM®)[8], to the Federal Aviation Administration (FAA) Integrated Capability Maturity Model (FAA-iCMM®)[9], to hybrid models (i.e., combining elements of two or more different models), to no identified model at all. There is no one right answer, but instead a variety of approaches are being tested by the small but growing DoD-wide software acquisition process improvement community of practice[10].

"... it is imperative that DoD program managers understand that their efforts will be measured against Section 804 requirements."

A new tool will soon be available to help those looking for acquisition best practices. The CMMI Steering Group, co-chaired by the DoD and industry, has sponsored the development of a CMMI-based guide for acquisition programs. The CMMI Module for Acquisition[11] focuses on effective acquisition practices used by first-level acquisition projects (e.g., system project offices/program managers). It also provides guidance to acquisition organizations above the acquisition project level to support institutionalization of those acquisition practices. In addition to covering the 804 requirements, many of the acquisition practices and amplifications in the Module are drawn from existing sources of best practices including the SA-CMM, the CMMI, the FAA-iCMM, as well as additional coverage areas defined by experienced acquisition professionals.

NAVAIR's Approach

As a key participant in the Naval Air Systems Command's (NAVAIR's) software acquisition process improvement program, the author is able to share with readers NAVAIR's approach as one data point. That approach is divided into three phases: 1) requirements determination, 2) gap analysis and planning, and 3) implementation, as explained below.

The requirements phase began by forming a small, command-endorsed team. That team selected relevant best practice models, mapped existing command policies to those practices, and developed and implemented a communications plan. The team chose a hybrid improvement model for mapping policies to practices. For pre-contract process areas[12], it selected the SA-CMM and for post-contract process areas, it identified the CMMI[13]. The team also added a ninth process area (to the eight provided by the OSD) – Measurement and Analysis from the CMMI – in order to emphasize the importance at NAVAIR of performance measurement.

The next phase entailed performing a policy gap analysis and developing a command-wide improvement plan. Policy owners identified changes to policy needed to comply with the selected best practices. A broader team was then formed – with representation from all executive program offices – to develop a NAVAIR software acquisition improvement plan (SAPIP). In addition, an existing SPI enterprise team, the NAVAIR Software Resource Center (SRC), was tasked to build or identify the infrastructure to support the SAPIP through a network of strategic partners.

Phase three was simply stated but represents a significant, long-term commitment: program managers execute the SAPIP and comply with revised NAVAIR policies. During the ongoing implementation phase, the SRC will work with individual programs to help them select the best practice model(s) that best support their business goals and baseline their processes.

Conclusions

Section 804's mandate for the DoD software acquisition process improvement programs is here to stay. It is not a one-time legislation with little or no follow-up, but the result of a consistent, well documented, and growing need. Already, congressional sources are considering actively

[SM] IDEAL is a service mark of Carnegie Mellon University.
® CMMI is a registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

identifying certain key programs for greater scrutiny to see if they have adequately implemented the requirements of the legislation. According to GAO sources, "The outcome is what's important, not which best practice improvement model is used as a road map to achieve the mandated requirements."

Given that the GAO and Congress feel that the acquisition of systems with major software components needs to be improved, it is imperative that DoD program managers understand that their efforts will be measured against Section 804 requirements.

As members of the DoD community, Section 804 is our collective call to action. While some DoD components and agencies have already taken steps to improve their software acquisition processes, others have not. NAVAIR, for example, has been addressing software development process improvement issues well in advance of Section 804 through an existing framework of system/software leadership teams. With the signing of Section 804, NAVAIR emphasizes its strategic goal to improve its software acquisition performance, continue to focus resources on refining policy, communicate implementation guidance, and expand its SPI support infrastructure. To achieve its goal, NAVAIR understood that top management support and metrics to gauge implementation effectiveness were essential.

How will your organization satisfy this critical need to improve?◆

References

1. U.S. General Accounting Office. "High Risk Series: An Update." GAO/HR-99-1. Washington, D.C.: GAO, Jan. 1999.
2. U.S. General Accounting Office. "Mission-Critical Systems: Defense Attempting to Address Major Software Problems." GAO/MTEC-93-3. Washington, D.C.: GAO, Dec. 1992.
3. U.S. General Accounting Office and National Security and Internal Affairs Division. "Weapons Acquisitions: A Rare Opportunity for Lasting Change." GAO/NSIAD-93-15. Washington, D.C.: GAO, Dec. 1992.
4. Jones, Capers. "Project Management Tools and Software Failures and Successes." CROSSTALK July 1998: 13.
5. Caterinicchia, Dan. "Firms Added to Army FCS Mix." Federal Computer Week June 2002.
6. U.S. General Accounting Office. "Defense Acquisitions: Assessment of Major Weapon Programs." GAO-03-476. Washington, D.C.: GAO, May 2003.
7. U.S. General Accounting Office and National Security and Internal Affairs Division. "Observations on the Department of Defense's Fiscal Year 1999 Performance Report and Fiscal Year 2001 Performance Plan." GAO/NSIAD-00-188r. Washington, D.C.: GAO, 30 June 2000.
8. Hayley, T., et al. "Raytheon Electronic Systems' Experience in Software Process Improvement." CMU/SEI-95-TR-017. Pittsburgh, PA: Software Engineering Institute, Nov. 1995.
9. Ferguson, Pat, et al. "Software Process Improvement Works!" CMU/SEI-99-TR-027. Pittsburgh, PA: Software Engineering Institute, Nov. 1999.
10. U.S. General Accounting Office. "DoD Information Technology: Software and Systems Process Improvement Programs Vary in Use of Best Practices." GAO-01-116. Washington, D.C.: GAO, Mar. 2001: 12.
11. Public Law PL-107-314[6].
12. Senate Report S.2514.

Notes

1. Defense Science Board Task Force on Defense Software, Nov. 2000.
2. The C Statement count is based on the average number of C Statements found within the typical function point.
3. The Independent Expert Program Review Working Group, established in Jan. 2001.
4. Software Intensive Systems Steering Group, chartered in Sept. 2000.
5. Additional process areas included configuration management, test and evaluation, integrated team management, and solicitation and source selection.
6. Information on the IDEAL Model can be found at <www.sei.cmu.edu/ideal>.
7. Information on the CMMI can be found at <www.sei.cmu.edu/cmmi/cmmi.html>.
8. Information on the SA-CMM can be found at <www.sei.cmu.edu/arm/ SA-CMM.html>.
9. Information on the FAA-iCMM can be found at <www1.faa.gov/aio/ProcessEngr/iCMM/index.htm>.
10. See the OSD's Acquisition Community Connection Web site at <www.acq.osd.mil>.
11. At the time this article was printed, the CMMI Module for Acquisition was pending publication. Its document number will be CMU/SEI-2004-TR-001.
12. Acquisition planning, and solicitation and source selection.
13. Requirements development and management, configuration management, risk management, project management and oversight, test and evaluation, and integrated team management.

Additional Reading

1. Defense Science Board reports can be found at <www.acq.osd.mil/dsb/reports.htm>.
2. General Accounting Office reports can be found at <www.gao.gov>.
3. Software Engineering Institute reports and other publications can be found at <www.sei.cmu.edu/publications/search.html>.
4. Back issues of CROSSTALK can be found at <www.stsc.hill.af.mil/crosstalk>.
5. A search function and online archive for Federal Computer Week can be found at <www.fcw.com/online archive.asp>.
6. Section 804 of Public Law PL 107-314 and other related documents can be found at the STSC Web site at <www.stsc.hill.af.mil>. Enter *Section 804* into the search engine.
7. You can search the Congressional Record at <www.senate.gov/page layout/legislative/d_three_sections_with_teasers/congrecord.htm>.

About the Author

Lisa Pracchia is a member of the Naval Air Systems Command's Software Resource Center. Her software background includes process improvement, business analysis, project management, product life-cycle management, and product marketing in a wide range of industries (discrete product manufacturing, international publishing, telecommunications, and defense). Pracchia has a master's degree in management from the University of Redlands.

Commander, NAWCWD
41K300D (L. Pracchia)
BLDG. 1494, STOP 6308
1 Administration CIR
China Lake, CA 93555
Phone: (760) 939-2188
DSN: 437-2188
E-mail: lisa.pracchia@navy.mil

Letter to the Editor

Dear CrossTalk Editor,

In my experience, CrossTalk is the best practical software development journal bar none. I have personally found it useful on many occasions, and assign it as reading for my teams.

In the December 2003 issue of CrossTalk, Barry Boehm is absolutely correct in "People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods" in that people are the most important factors to success. My personal experience with agile methods leads me to strongly concur in valuing individuals and interactions over processes and tools. However, picking the right people is not always an option. Too often in either government or contractor shops, the front-line team leader has little choice regarding team membership – regardless of how well the current pool of talent matches the new task – because the first task is always job security for existing employees. My experience is that in such situations, success is average, but true excellence is hard to come by.

I also liked Dennis Linscomb's article "Requirements Engineering Maturity in the CMMI," also in December's CrossTalk. He has in me a kindred spirit in regards to the poor state of affairs in requirements engineering. He has an excellent idea with his requirements engineering maturity levels, but I disagree that Capability Maturity Model® Integration (CMMI®) has the cart before the horse in putting management before technical execution. CMMI is good at telling us *what* but less good at telling us *how*, and even worse at telling us how to get from where we are to where we need to be. This was one of my first revelations about CMMI.

In deciding to implement CMMI, the first thing an organization has to do is figure out and write down what they are doing in each process area. The second thing to do is figure out where the organization needs to go. The third thing is how to get there. Success is achieved one step at a time, one change at a time. Once the change process is in place, the organization can work on optimizing technical performance. Individuals who have the shirt sleeve, dirty-fingernail knowledge of how to implement specific best practice techniques in the day-to-day work environment are worth their weight in gold. Few people can give you tips on precisely what best practices to implement in getting to Level 2 and higher. When you find such people, pay them a lot to keep working for you.

Ralph Nebiker
ENWGS Modernization
SPAWAR Systems Center
nebiker@spawar.navy.mil

CrossTalk *invites readers to submit their thoughts, comments, and ideas on its themes and articles as a "Letter to the Editor." Simply e-mail letters to* <crosstalk.staff@hill.af.mil>.

Web Sites

Defense Procurement and Acquisition Policy

www.acq.osd.mil/dpap

The Defense Procurement and Acquisition Policy Web site is a complete source for Department of Defense acquisition information. It features the latest news and events, print and electronic publications, a Knowledge Management page of frequently asked questions, the "Defense Federal Acquisition Regulations Supplement," workforce training and career development information, and more.

Procedures for the Acquisition and Management of Technical Data

www.dtic.mil/whs/directives/corres/pdf/501012m_0593/p501012m.pdf

The Procedures for the Acquisition and Management of Technical Data is the official manual prescribing policies and procedures for the Department of Defense's acquisition and management of technical data.

FARSite

http://farsite.hill.af.mil

FARSite is the Federal Acquisition Regulation (FAR) information site sponsored by the Contracting Laboratory at Hill Air Force Base. The FAR is the primary regulation for use by all federal executive agencies in their acquisition of supplies and services with appropriated funds. The regulation is published on FARSite in addition to supplements for the defense department, Army, Air Force, Navy, Marine Corps, special operations, and NASA.

ASSIST-Quick Search

http://assist1.daps.dla.mil/quicksearch

ASSIST-Quick Search provides direct access to Department of Defense (DoD) and federal specifications and standards available in the official DoD repository, the Acquisition, Streamlining, and Standardization Information System (ASSIST) database. The ASSIST-Quick Search locates documents available for distribution by the DoD Single Stock Point for Military Specifications, Standards, and Related Publications. Retrievable data includes military standards, specifications, data item descriptions, and more.

Acquisition Management Systems and Data Requirements Control List

www.dtic.mil/whs/directives/corres/html/501012l.htm

The Defense Technical Information Center (DTIC) is host for the Department of Defense (DoD) 5010.12-L "Acquisition Management Systems and Data Requirements Control List" soon to be published on this Web site. The DTIC is the central facility for collecting and disseminating scientific and technical information for the DoD. The DTIC serves as a vital link in the transfer of information among DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors.

Why We Need Empirical Information on Best Practices

Dr. Richard Turner
The George Washington University

Best practices are widely recommended as a way to improve organizational performance, especially in software-related endeavors. This article takes a skeptical view of the current way best practices are identified and prescribed. It identifies relevant information that is often missing from best practice discussions and recommends an alternative approach to gathering, evaluating, and applying that information.

In the history of software development and acquisition, one of the most often prescribed curatives for their continuing infirmities, aches, and agues is the identification and implementation of *best practices*. Of course, the notion of what defines a best practice is not clear. Some best practices, for example configuration or risk management, are actually disciplines seen as crucial to success. Other best practices are broad approaches or philosophies such as architecture-first development or Integrated Product and Process Development. A third type of best practices, peer reviews for example, are actually practices proven to be beneficial in a specific way. In reality, the term has been so broadly applied as to be nearly meaningless.

In spite of being *definitionally challenged*, best practices continue to arise – sometimes as ephemeral answers du jour and other times as lasting wisdom. They populate the lists and fill the books that we turn to for guidance. Unfortunately, we all too often find that the benefit is more in the eyes of the beholder than in any measurable result of implementing the enshrined practices. We ultimately do not know, beyond anecdotes and sales pitches, whether a practice will work for us. So, to move from faith toward science, we need to approach best practices in a skeptical but constructive manner. I believe that the best way to do this is through focused empirical studies and careful analysis that result in a validated assessment of the practice's cost and real benefit.

Some History

My earlier research into the adoption of best practices in defense acquisitions uncovered considerable recognition of the most widely referenced best practices, but very little real implementation [1]. There were good reasons for the unsuccessful implementation of even the highly recommended practices, and most had to do with lack of information.

I found that practices – best or otherwise – generally do not fall into the *one-size-fits-all* category, and it is not easy to evaluate how appropriate a practice is for a particular organization or program. Most practices also have hidden assumptions and conditions for use, and there is little available support for evaluation and selection. When a practice is chosen, there is often little information on how to implement it in the real world. Consequently, managers often find themselves acting on a *faith-and-gut* feel in deciding what practices to implement.

There are also the instances of best practices that are not. A case in point is the venerated heuristic that the larger a software module, the more likely it is to have defects. Surprisingly, empirical study at NASA's Software Engineering Laboratory showed that smaller modules actually increased the defect rate for a period, and that there existed a *sweet spot* where the module size corresponded to the fewest defects. The exact placement of the sweet spot depends on a number of characteristics about the software being developed, but Figure 1 illustrates the general finding, which is in direct conflict with the previously held *best practice*.

> *"... empirical study at NASA's Software Engineering Laboratory showed that smaller modules actually increased the defect rate for a period ..."*

Applying Empiricism: Questions Needing Answers

Empiricism, in this context, can be thought of as a methodical approach to the gathering and analysis of data about a specific practice. It is applying, to the best of our ability, scientific principals to the evaluation and validation of practices with the goal of producing usable information. This is more than collecting anecdotes or drawing general conclusions from a few unstructured experiences.

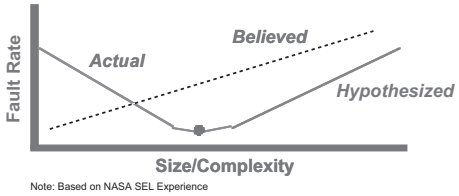
Empiricism should not be confused with quantitative analysis, since there are ways to methodically collect and meaningfully analyze qualitative data. Quantitative data is certainly a worthy goal, but in some cases it is very difficult to obtain. For that reason, we include several qualitative approaches, including workshops and expert opinion, under our empirical umbrella.

The primary purpose of methodical analysis of practices is to gather and maintain data to answer specific questions. Using this data, including knowledge gained from lessons learned in actually implementing practices, we can build tools to help select practices that are appropriate for a particular project. Let us look briefly at some questions to which empiricism can provide validated answers.

How Much Will It Really Cost?

It is usually risky to order from a menu without prices, so the first thing we need to know is the size of the bill. Let us consider some of the major costs we need to define and capture. How many hours of training are needed? Are there tools or other infrastructure required? Of course, these upfront costs might just be the tip of the iceberg. What are the costs of the effort and resources needed to actually apply the practice? Are there license fees or equipment maintenance associated with the infrastructure? Unexpected costs

Figure 1: *Notional Findings on Module Size versus Fault Rate*

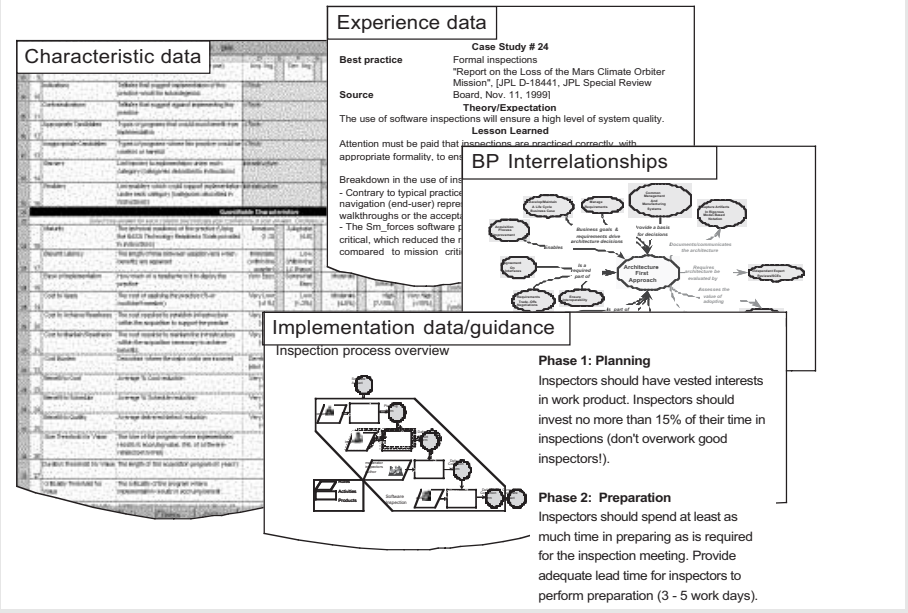


The Department of Defense Best Practices Clearinghouse — First Steps Toward Empiricism

The Office of the Under Secretary of Defense (Acquisition, Technology and Logistics) Defense Systems has initiated an activity to define an empirically-informed clearinghouse for software acquisition and development best practices. The Data Acquisition Center for Software (DACS), the Fraunhofer Center at the University of Maryland, and the Center for Software Engineering at The University of Southern California are specifying the infrastructure and processes required for a centralized, empirically-based resource for acquisition and development projects as shown in the Conceptual Best Practices Clearinghouse Data figure, below.

The clearinghouse is envisioned to maintain validated practice information, support user-driven selection of practices, provide step-wise implementation guidance, and track implementation results. Easy-to-use, informative tools will suggest appropriate practices based on goals, risks, life-cycle phase, and program environment. Support for evolving from basic to advanced practices could also be included. Web-based access tailored to user needs is planned, as well as an active infrastructure to link expertise and information providers to users via communities of practice, courses, workshops, publications, and shared pilot projects.

A user advisory group is being established to ensure that the products and tools to be provided will meet the needs of developers and acquirers. The clearinghouse team is seeking submission of best practices, implementation and results data, and lessons learned from development and acquisition organizations. Coordination with service and agency best practice and lessons learned repositories is underway. For more information, contact Kathleen Dangle of the Fraunhofer Institute at <kdangle@fc-md.umd.edu>, or Tom McGibbon of the DACS at <tom.mcgibbon@itt.com>.



can doom any benefit that might be achieved. Maintaining information on how much a practice costs to implement is a key empirical characteristic.

What Is the Actual Benefit?

OK, we have an idea of the cost but really, how good is that best practice entree? What exactly do we expect from implementing the practice? Will it shorten the schedule, raise quality, or lower cost? If so, by how much? What specific risks could it mitigate? How are benefits measured? Sometimes there are hidden benefits or ones that surface late in the life cycle. On the other hand, even obvious benefits might need actions outside of the project's control to be fully realized.

For example, peer programming can provide higher quality and shorter development times, but successful implementation might require changes to corporate policy regarding reward structure, office space, or equipment allocations. Validating the benefit ensures that recommended practices have a fighting chance of helping programs that implement them. Benefits may not be explicitly captured in dollars, but the type, nature, and magnitude can be collected and analyzed.

What Is the Pedigree?

It is always good to know where the practice came from and who actually established it as a best practice. Is it technologically mature? Are there studies that suggest it works? Who has successfully implemented it? This is especially true when proprietary components such as tools or processes are part of the practice. *Caveat emptor* is a pretty good mantra for our empirical activities. Data on the number of implementations, breadth of application, and the level of consensus on the practice's value by experts are means to address pedigree empirically.

Is the Environment a Critical Success Factor?

Every practice does not apply to every type of project. Does the practice assume a particular type of development or acquisition environment? Does it only work for small projects? Was the best practice identified in an environment of stable requirements or is its primary benefit only realized in a situation of constant change? When in the product life cycle is it best applied? It might not be helpful to implement a requirements practice when the program is knee-deep in integration testing. What is the size or criticality threshold at which the practice begins to pay off? What is reasonable for the F-35 or a Missile Defense Agency component might not be appropriate for a commercial off-the-shelf-based, Web-enabled training application. Maintaining the characteristics of the environments where a practice has been implemented and the associated results is one way to capture this data.

How Long Before It Works?

Knowing the time it takes for a benefit to be realized is one of the subtlest questions to answer. Does the practice provide immediate benefit, or do its effects have to trickle down through the development or acquisition process for months (or years) before actually helping? Compare the benefit latency of peer reviews with that of a process improvement program. The first pays dividends immediately while the second takes months to show measurable value. Knowing the benefit latency also has an unfortunate down side

– if it will not help by the end of someone's watch, it may be more difficult to gain support for implementation. Benefit latency can be characterized based on lessons learned and experience reports.

Are There Other Barriers?

Practices are implemented by people, so they imply change. The project team's attitude, capabilities, and personality can raise all sorts of problems. Will they accept and adopt the practice or just go through the motions? As with any change, corporate culture also plays a part. Will management buy in or fight it every step of the way? How will the practice impact the organizational infrastructure? Knowing what barriers have historically manifested is a major advantage in planning successful implementation. Barriers can be identified from experience, rated as to impact, and organized into useful categories.

Can We Implement This?

Finally, we need to understand the probability of successfully bringing the practice to our particular program. Are the existing resources and authority sufficient to implement the practice? It is usually possible to implement something that affects the way a team works internally, but implementing something like Integrated Product and Process Development with all of the significant impacts on other stakeholders requires enormous resources and power. Is there sufficient time left in the project to achieve any benefit? Clear instructions as to how to implement the practice are also priceless. Knowing about available tools or consultants or classes can save the effort of making it up as you go.

There has to be an honest assessment of the implementation requirements and the ability to meet them, or its likely implementation will be incomplete or shoddy, and the project possibly worse off than before. Capturing the scope of control and other requirements for implementation is relatively straightforward and can support implementation guideline development.

Conclusions

You probably recognized that answering these questions could be extremely difficult. It will take a focused, ongoing effort to gather and maintain the data required to validate the effectiveness and costs of practices. This will be ongoing because the data, as well as the practices, will change continuously over time. We know that every practice has associated cost and benefit, maturity and pedigree, preferred environment, benefit latency, organizational barriers, and required competencies for successful implementation. We need to seize the opportunity to capture, analyze, and package the precious information of others' experiences. There is so much knowledge and experience being gained daily by Department of Defense programs that it is a travesty to let it simply vanish when the technology exists to make it useful and available.

Consider the impact to projects of a successful effort to empirically gather data and validate best practices. How marvelous to be able to pick vetted, proven practices that apply to our particular needs and resources, reasonably confident that the implementation will bring about predictable benefits. The reduction of rework and wasted effort could well dwarf the expenses associated with the validation effort. Above all, projects would have another means to increase their probability of success in an environment that has seen all too many failures.◆

Reference

1. Turner, Dr. Richard. "A Study of Best Practice Adoption by Defense Acquisition Programs." CROSSTALK, May 2002: 4-8.

About the Author

Richard Turner, D.Sc., is a research professor in Engineering Management and Systems Engineering at The George Washington University. He is currently supporting the Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, Defense Systems. Turner is a co-author of "CMMI Distilled" and "Balancing Agility and Discipline: A Guide for the Perplexed." Turner has a Bachelor of Arts in mathematics from Huntingdon College, a Master of Science in computer science from the University of Louisiana at Lafayette, and a Doctor of Science from George Washington University.

George Washington University
1931 Jefferson Davis HWY
STE 104
Arlington, VA 22202
Phone: (703) 602-0851 ext. 124
E-mail: rich.turner.ctr@osd.mil

Coming Events

April 19-22
2004 Systems and Software Technology Conference
Salt Lake City, UT
www.stc-online.org

May 11-13
Technet International
Washington, DC
www.technet2004.org

May 17-21
STAREAST
Orlando, FL
www.sqe.com/stareast/

May 23-28
26th International Conference on Software Engineering
Edinburgh, Scotland
www.jupiterevents.com

June 2-4
Sacmat 2004
Yorktown Heights, NY
www.sacmat.org

June 11-13
ACM Sigplan 2004 Conference on Language Compilers and Tools for Embedded Systems
Washington, DC
http://lctes04.flux.utah.edu

June 23-26
Agile Development Conference 2004
Salt Lake City, UT
www.agiledevelopmentconference.com

June 27- July 2
USENIX Annual Technical Conference
Boston, MA
www.usenix.org/events/usenix04

A Project Risk Metric

Robert W. Ferguson
Software Engineering Institute

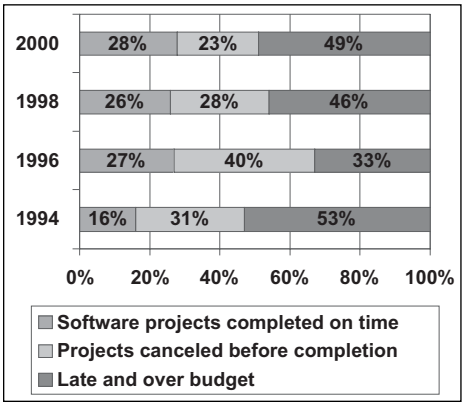
A risk metric is proposed that is normalized across projects. The purpose of the metric is to provide management visibility into project uncertainty. This works best in an organization that manages multiple projects. The proposed metric can be applied early and throughout the project. It has been useful for identifying or canceling projects in trouble. It has also been useful for identifying projects that do not yet have a satisfactory risk plan.

The Standish Group published its original "Chaos Report" [1] in 1994 declaring that American companies spent $81 billion on cancelled projects. Additional Standish Group data in Figure 1 shows that the situation has not improved as much as one would hope.

Even projects that are not cancelled may deliver such reduced functionality that most people would not count them as successful projects. Often there has been early evidence that the project was headed for disaster. The project manager may even have issued warnings to senior management or sponsors about the problems. There simply seemed to be no way to *pull the plug* until the project was already over budget, late, and at the point where the customer was ready to give up or worse.

The problem may be a failure to examine the risks of the project from a systemic view. When risks are faced one problem at a time, the management team may convince themselves that every problem can be addressed, or that each problem has a low probability of occurrence. However, the collected problems may still be too much to manage. By its very nature, risk is statistical. It is possible to examine the collection of risks and make some projections about the project's likely success or failure. The result can even suggest that certain projects should be cancelled very early. Such projects can be rescoped and rebudgeted in a way that improves the focus and likelihood of success.

Figure 1: *Standish Group Project Results*



Risk Management Process

The Project Management Body of Knowledge (PMBOK) [2] includes a chapter on risk management. It describes the process steps as follows:

1. Risk Planning.
2. Risk Identification.
3. Qualitative Risk Analysis.
4. Quantitative Risk Analysis.
5. Risk Response Planning.
6. Risk Monitoring and Control.

"A standard definition of risk is an uncertain event that would cause an uncertain impact on project schedule, cost, or quality."

The metric proposed in this article fits the Qualitative Risk Analysis stage so it can be used as early as possible throughout the project duration. Rough estimates are available at this step, and are sufficient for an assessment of the overall project risk. However, the rough estimates will not suffice for risk items requiring real risk-response strategies such as mitigation and avoidance plans where more detailed work is needed.

In this metric, the distinction between steps three and four of the process model is important. The metric supports the viewpoint of senior management who wants to determine which of several projects has significant uncertainty. The project itself must deal with specific risks and quantitative analysis. As such, a risk manager on a large project will not find this metric as useful. He or she must have much more specific information.

History and Metric Definition

Risk is both old and new. The written history of risk begins in 1491 with the "Pacioli Puzzle," which arises from gambling when the game is stopped before completion [3]. The problem was solved by Pascal and Fermat in 1654 and so began the use of risk in forecasting. Today, risk is the core concept in insurance and has become a major focus in project management.

A standard definition of risk is an uncertain event that would cause an uncertain impact on project schedule, cost, or quality. Both the event and the impact have the element of uncertainty. The definition from probability theory is a bit more restrictive but it provides us with the metric:

R = P x V

The metric value of risk (R) is the product of the probability (P) of the event with the most likely value (V) of the outcome. If the risks are independent, we can add these estimates together for a combined estimate. So overall project risk is the sum of the separate risks.

Total Risk = Sum of all (P x V)

The Total Risk value and trends of Total Risk provide a picture of the project, making it easy for people to see some good and bad project patterns without delving into the statistical theory. The assumption about independence is necessary for the theory. However, in practice, risk management experts are aware that risks are not always independent. The metric is based on the theory derived from gambling where the assumption holds true.

Getting the Probability

There have been a few sociological studies showing the range of errors people demonstrate in estimating risk. Choosing an appropriate range helps when no historical data is available. Table 1 and its heuristics have been useful in avoiding the problems of underestimating and overestimating risk. Remember, most project managers see only three to five projects in their

career at any one company so they work from a very restricted sample. They need heuristics for estimates.

Five levels of probability seem to work well. Colleagues have not had a problem assigning an event to one of the recommended levels, so the suggested ranges provide good separation.

Label	Description	Value
Very Low	In your career, you have never seen this happen, but it could.	5% Range 1-9%
Low	It has happened on occasion.	25% Range 10-29%
Moderate	Sometimes it happens and sometimes not.	50% Range 30-69%
High	Most of the time this event will occur.	75% Range 70-89%
Very High	It has happened on every project, and you think it always will, but there is a chance of escape.	95% Range 90-99%

Table 1: *Risk Event Probability Estimates*

Analyzing the Impact

The impact of a project risk-event needs to be similarly divided into a few classifications and assigned a numeric value to manage risk. Making the numbers match conceptually when one risk affects schedule, another cost, and another quality or scope can be a bit of a stretch so a method is required to normalize the numbers.

A quick simplifying assumption works for the qualitative analysis stage: assign a single impact type. Choose from one of the following three image types: schedule, cost, or customers (sales). It is true that a risk event may affect more than one of these, however, coming up with a value for all the possible effects is challenging and probably not a worthwhile exercise until quantitative analysis. Narrowing the discussion of a risk event to a single type impact also focuses attention on the most useful response plans. This approach helps to avoid the problem of overthinking the impact of a risk. Here is an example of the kind of thinking to avoid at this early stage.

> Some employees are due for sabbatical leaves of two months. One may take that sabbatical during the project. You propose that turnover is a risk for the team. If this risk event occurs, it may cost some additional schedule time and additional re-source cost to hire and train staff. If you lose schedule time, you may also lose some sales. What is the appropriate impact for this event – schedule, cost, or sales?

Experienced risk managers will understand that additional impacts will have to be considered when developing the risk-response plans.

Normalizing Risk Impact

The next challenge is normalizing the various impacts to arrive at a single numeric value for schedule, cost, and sales. Capers Jones reported that in 1996 "the typical project is 100 percent over budget when it is cancelled" [4]. This suggests that a useful normalizing factor is to set maximum risk impact at project cancellation. That impact value should be cost or schedule overrun of 100 percent, or when there is no customer or no potential first-year sale.

Of course, no project will be allowed to overrun to such an extent without senior management intervention, but that is precisely the point. Senior management should intervene when the uncertainty suggests the project is in trouble. Since the metric is applied at qualitative analysis, there is time to recover.

A Second Aside

Why would we develop a product without customers? No one plans a project for a non-existent market, but the market can disappear or be misjudged. It happens all the time. Some well-known examples are the Newton tablet computer, the Iridium satellite telephone, and New Coke. Everyone also has an example of the *pet project* that was developed but was never used. Many organizations are surprised to learn that it is possible to cancel a project when sales or number of customers are factored into the risk management effort.

Using the possibility of cancellation as the highest risk impact, assign a value of five to cancellation. Five levels of risk should be enough. Creating the other levels again requires a bit of psychology. The PMBOK states an order of magnitude estimate is plus or minus (±) 35 percent of the base estimate. Using a range of 1 ± 0.35 is a range of 0.65 to 1.35. The ratio of these two numbers is 1.35/0.65 = 2.08, approximately a factor of two.

Thus to have a range that clearly separates the estimates, we must use a larger value. Using ±50 percent yields a ratio of 1.5/0.5, which equals a factor of three.

Experience suggests the psychology works, and people are comfortable with the results. Therefore, assign five to cancellation and divide the cancellation level by three successively to arrive at the other values. The following example points the way.

Consider a project that is scheduled for 18 months with a projected cost of $30 million and projected first-year sales of $27 million. This would be a project of about 100 people with about $5 million in external expenses. A risk event with an impact level of five would cause the following:

- Overrun by 18 months.
- Overspend by $30 million.
- Achieve no first-year revenue.

A risk event with an impact level of four (divide by three) would cause the following:

- Overrun by six months.
- Overspend by $10 million.
- Lose $9 million in sales (achieves $18 million).

A risk event with an impact level of three would cause the following:

- Overrun by two months.
- Overspend by $3.3 million.
- Lose $3 million in sales.

A risk event with an impact level of two would cause the following:

- Overrun by three weeks.
- Overspend by $1.1 million.
- Lose $1 million in sales (one customer).

A risk event with an impact level of one would cause the following:

- Overrun by one week.
- Overspend by $300,000.
- Lose $300,000 in sales (customer delays six months).

A useful interpretation is to say that the project manager can manage one or two risk events of impact level one within the project contingency and without unusual reporting. It would be necessary to generate a special report for any occurrence of impact level two. Any risk event at impact levels three or higher will require senior management's involvement to determine the response.

There is one more step in calculating the final impact. The numbers one through five were calculated by successive division by three. The final value has to put that back into a geometric scale.

Impact = 3^(level-1)

Impact Level	Impact Value
5	81
4	27
3	9
2	3
1	1

Table 2: *Impact Value Adjustment*

So a risk event with an impact level of five has an impact value of 3x3x3x3=81, as shown in Table 2.

The factor *three* is not arbitrary but is derived from the observation that order-of-magnitude estimates use a factor of two for the error range.

There is a temptation to turn the numbers back into dollars. This is a lot of work as revenue dollars are not the same as cost dollars or schedule days. The extra work makes sense for the top risks but not in general. Using the impact number instead of a dollar value also normalizes the risk metric across projects.

Risk Calculation

The final risk calculation follows the original equation:

Risk = Probability x Impact Value

The highest risk is 95% x 81 = 76.95
The lowest is 5% x 1 = 0.05

Normal usage is the sum of the highest 20 project risks. It seems that 20 risks at a time is a sufficient number to track for all but some mega-projects (over three years and more than 500 people). Barry Boehm, TRW professor of software engineering at the University of Southern California and author of the COCOMO estimating model, has suggested that projects manage the top 10 risks. There are two reasons this metric recommends watching the top 20. The first is that 20 risks x 5% = 100%. That is the recommended cancellation level so it makes for a convenient metric. The second reason is to make certain the project team investigates more than the first 10 risks to be certain that it manages the top 10.

Project Risk Score =
Sum (highest 20 project risks)

Implications

The Project Risk Score should be charted so senior management can see scale and trends. Since there is a threshold (threatened cancellation) implicit in a risk with impact level five, that threshold should also appear on the chart. An impact level equal to five translates into an impact value of 81. Figure 2 is a sample chart from an actual project.

There are many implications in the chart and its use. The threshold is a powerful concept. Senior management will focus a lot of attention on a project that is above the threshold. The fact is, projects with risk higher than the threshold simply will be late, over cost, or fail to meet project quality goals. Some projects have risk levels that are astronomically high. It is theoretically possible to see a value of 1,539 with 20 risks that are very likely to occur and have an impact rating of five. Of course, such a project should be cancelled and restarted. I have actually seen only one project risk value over 400. That project had to make major changes to deliver even a subset of the desired functionality. If the threshold concept had been introduced at the start of that project, it would never have gotten into so many problems.

A somewhat opposite situation also can occur when a project shows particularly low risk. The project manager or senior management may have a sense that a project is at significant risk, but the metric does not show it. Use that low number as a signal that a risk collection effort is needed. The project manager must gather a wider audience and run a facilitated session to identify those other risks. Make sure to include stakeholders from other locations and groups outside the development team. Develop the organization taxonomy for risks like the one in the "Continuous Risk Management Guidebook" [5] from the Software Engineering Institute to make the data collection more complete and rigorous.

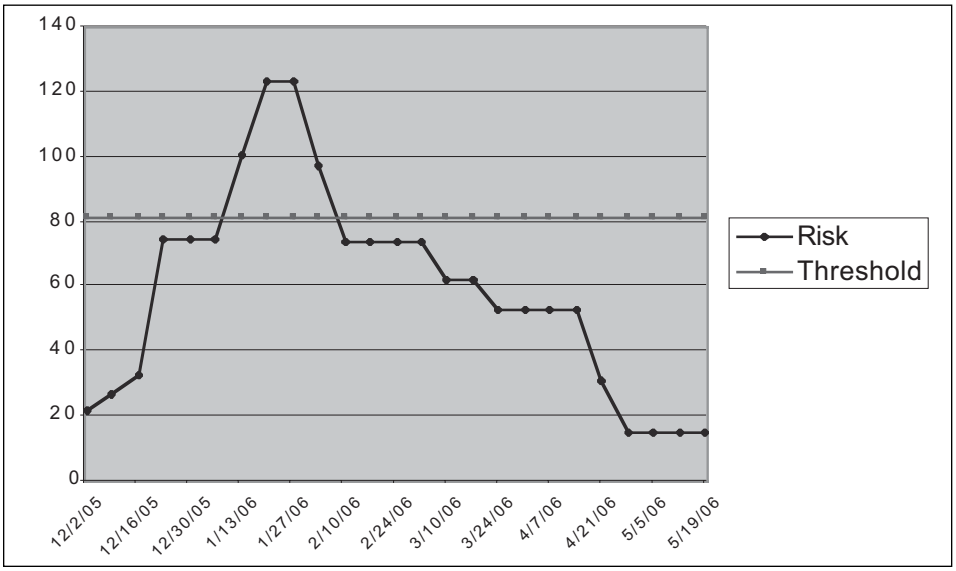
A normal response when the project risk is high is to manage that risk down. This can happen several ways:

- The time for the risk event may pass without incurring the problem.
- The team may adopt an avoidance plan so that the event cannot occur.
- The team may adopt a mitigation plan to reduce the impact.
- The team may transfer the risk to another organization.
- The event may occur and the project eats the contingency.

The last four responses cause the project to incur a specific cost that should appear in the project planning and reporting. Each of the responses requires the project manager to make some update to the risk database.

Finally, product managers (not project managers) should be hesitant to select a project of very low risk. If the risk is so low, why not address a more aggressive product plan? Risk avoidance is not generally a winning strategy in the marketplace. The point is to manage risk to appropriate levels for the organization, product, and project. Risk management is a systemic study and not a technological one.

Figure 2: *Sample Project Risk Score*



Implementation

There are several challenges in adopting the project risk metric. The following is a list of the top challenges:

- **A database for collecting and managing risks.** There are a number of products that will do the job. Implementing one will require the addition of project and sub-project identification and organizational process support. This work cannot be institutionalized without an automated system.
- **A process model.** The basic framework is available in the PMBOK, the Continuous Risk Management Guidebook, or the Institute of Electrical and Electronic Engineers standard for risk management [6]. The process model has to be extended to cover a risk taxonomy that is appropriate to

the organization.

- **Automated reporting.** Chances of success are better if the project risk chart is automated and is required as a part of the regular project management review. The risk metric should be checked at least monthly.
- **Training.** Training is a big effort. Training project managers to do risk management takes days, not hours. Writing good risk scenarios requires at least eight hours of training and much practice. Learning the organization taxonomy of risk takes time. Evaluating impacts probably takes three hours of training. Directors and senior managers also need at least three hours of training. Do not attempt to implement a project risk metric without decent training on risk management.

Summary

Many seasoned project managers say that advanced project management is mostly risk management. This metric makes that statement visible and concrete to a much larger audience. It provides fast visibility and has a high emotional impact on managers.

The project risk metric, however, has been tested in only one location and on only a dozen projects. The simplifying assumptions made in order to develop and use the metric make it suspect for use by risk practitioners who must perform detailed quantitative analyses and develop risk mitigation and avoidance plans.

It does provide a comparison between projects that is useful to senior management. If senior management is presented with one risk at a time, they are likely to develop a confidence that they can deal with each risk as it comes. Dealing with each risk separately and successfully may convince them that the project cannot really be in trouble. Management may then come to believe that the project team is whining about problems instead of dealing with problems, and real risks may not be addressed in a timely fashion. Presenting senior management with a picture of the total project risk will encourage them to take appropriate systemic actions when these are necessary. Product managers on projects with high risk will need additional justification and resources to add scope. The development team may have an easier time getting training or adding consultants when needed.

The key is presenting senior management with better visibility into the project so that project change-management becomes faster and easier, and finally, so that product delivery becomes predictable.◆

References

1. The Standish Group International, Inc. CHAOS Chronicles Ver. III. West Yarmouth, MA: The Standish Group, 2003 <www.standishgroup.com/chaos-resources/chronicles. php>.
2. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). Newton Square, PA: Project Management Institute, 1996 <www.pmibookstore.org/productdetail.asp?productid= 4106>.
3. Bernstein, Peter L. Against the Gods: The Remarkable Story of Risk. Hoboken, NJ: John Wiley and Sons, 31 Aug. 1998 <www.wiley.com/Wiley CDA/WileyTitle/productCd-0471295639.html>.
4. Jones, Capers. Patterns of Software Failure and Success. Boston, MA: International Thompson Computer Press, 1996.
5. Durofee, Audrey J., et al. Continuous Risk Management Guidebook. Pittsburgh, PA: Software Engineering Institute, 1996 <www.sei.cmu.edu/publications/books/other-books/crmguidebk.html>.
6. Institute of Electrical and Electronic Engineers. "Software Engineering: Soft-ware Life-Cycle Processes, Risk Management." Proposed Standard. New York: IEEE, 2004 <http://standards.ieee.org/announcements/pr_p1540. html>.

About the Author

Robert W. Ferguson is a member of the Technical Staff at the Software Engineering Institute. He has more than 30 years of software development and management experience in several industries. Ferguson is a member of the Computer Society of the Institute of Electrical and Electronic Engineers and the Project Management Institute. He has been active in the software process improvement community for several years and is past chairman of the Chicago Software Process Improvement Network.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-9750

Agile Software Development for an Agile Force

John S. Willison
U.S. Army CECOM Software Engineering Center

I remember in a meeting I attended, an Army general arguing the need for the software community to fall in-line, saying that "Software itself has never killed anyone." Maybe, maybe not, but I have seen software kill many Army programs and careers. As the Army transforms itself into a more agile force, the Army software community continues to struggle with the challenge of effectively providing software to support that force. This article identifies some components of an effective approach to software development and provides an example that is leading the way.

It is necessary to provide a characterization of the current U.S. Army business environment to set the context for the recommended components for good business. Historically, the Army acquisition and development processes have been driven by the attempt to institutionalize success and avoid failure. The Army management and acquisition processes are based primarily on hardware models that, in turn, are based on the value-added discipline of risk management.

With hardware, it is critical to mitigate risk and get it right the first time, particularly prior to entering any stage that involves significant expense such as production. The Army has evolved into using a rigid approach where requirements are defined and then used as the basis for development, testing, and determining success. Further, as development and hardware sustainment are different activities, the Army has defined different processes and funding strategies for these distinct activities.

With software, the processes and investment strategies are different than hardware; the risks are also different, and yet we attempt to manage them the same way. Software sustainment to a large degree is simply doing more development; however, development and sustainment are often managed by different organizations and funded differently. The risks associated with software are different as well; and yet, we attempt to manage them the same as we work through a sequential series of milestones. The real risks with software are in taking too long before giving the user something that knowingly will evolve over time, and in measuring success as meeting predefined requirements as opposed to getting the user something he or she wants and likes.

There are other factors that influence the way the Army acquires and develops software. For the increasing percentage of Army capabilities that are hosted on commercial off-the-shelf (COTS) hardware platforms, competition to provide software is expanding and the barrier to enter the competition is low. Users have access to a wide range of sources, and more importantly, a wide range of sources have access to users. Increasingly, initial and incremental capabilities can be provided to users as software-only releases, and in some cases simply can be downloaded over the network.

Finally, there is this question: how much alike or different should the Army software community be from the commercial software industry? Clearly there are some differences. Most notably, the Army software community's priority is capability and readiness whereas the commercial software industry's priority is profit. Those different priorities have historically been used to rationalize the need for unique and rigid approaches to software.

Components for Good Business

The Army develops, integrates, and employs as wide a range of software-based capabilities as any other organization. While no single method for improving the Army's approach to software development would suffice, there are some common components for improvement.

Balance Between Plan-Driven and Agile Development

The October 2002 edition of CROSSTALK [1] did an excellent job of contrasting the *plan-driven* [2, 3] and agile development approaches to software, and the spectrum between these two perceived extremes. There is much to be gained from both approaches.

The Software Engineering Institute's Capability Maturity Model®, for example, has done much to address software as an engineering discipline and the need for a plan-driven approach. Agile development, as characterized by the Agile Alliance [4], finds:

> … more value in individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan.

Historically, the Department of Defense and the Army have emphasized process. To produce a more agile force, the Army needs a software community that has process discipline, but is more agile as well.

Get as Close to the User as Possible

The only one who truly knows what the user wants or needs is the user himself. The closer you get to the user, the closer you will get to developing software that he or she will accept and adopt; it is never too early to do this.

Show Them, Ask Them, and Repeat Often

The Army is very good at generating requirements, and generating endless cycles of life-cycle events aimed at meeting those requirements. Instead of asking users what they need and then getting back to them only after developing the solution, the Army should be prepared to show users what they could get up-front. If nothing else, this builds the users' confidence that the Army is able to deliver something. The focus should be on early and continuous software delivery.

Architecture, Architecture, Architecture

Architect Frank Lloyd Wright is believed to have said that no matter what you are building, always remember:

> It will take longer than you plan, it will cost more than you figured, and it will be messier than you could have ever have anticipated. But remember the most important thing is not what is visible. What's

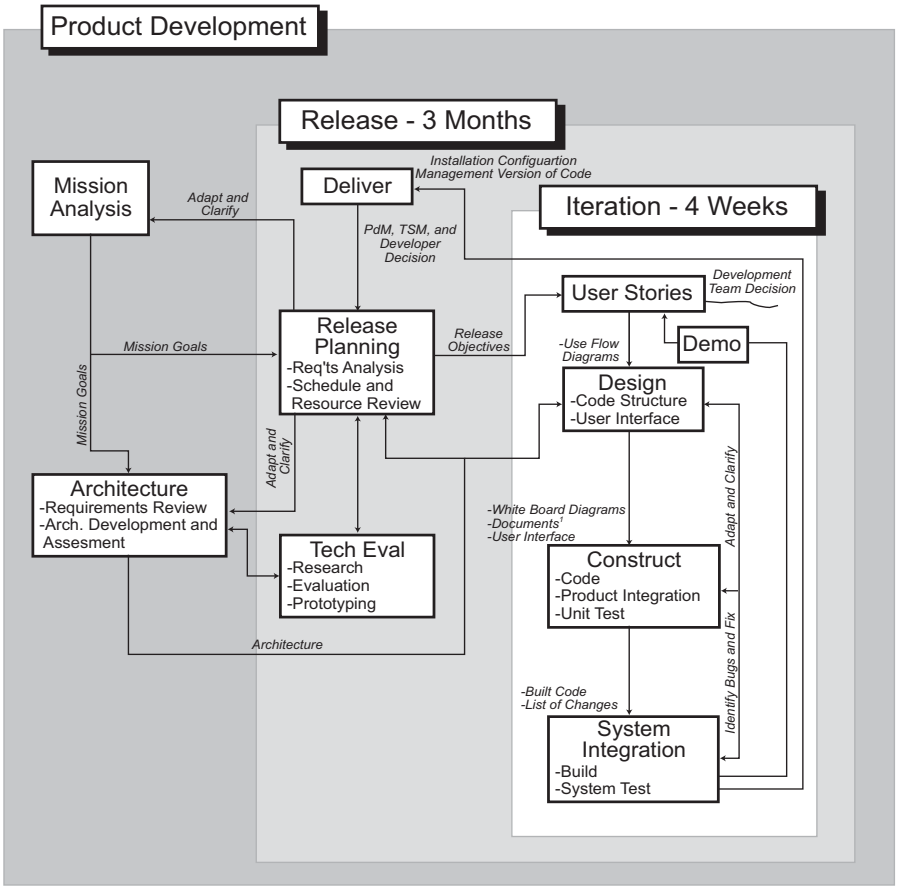
most important is the foundation.

Bad software products do not necessarily have bad software architectures. However, good software products are likely to have good software architectures. While the Army, and everyone else for that matter, has increased the amount of attention and discussion surrounding software architectures, the understanding and practices associated with software architectures are still not sufficient.

Senior Army leaders echoed the need for the Army's development approach to be more agile and responsive at a recent Association of the United States Army Symposium [6]. Lt. Gen. John Caldwell, military deputy to the assistant secretary of the Army (ASA) for Acquisition, Logistics, and Technology (ALT) said, "If you wait to put a perfect capability in the field, you will never put anything in the field," and, "People are the key to our business."

Col. Bruce Jette, director of the recently established Rapid Equipping Force, said that the goal is to go from "idea to equipping" in two to three months and that, "field commanders are tremendously accepting" of this approach.

Col. Nick Justice, director Future Force ASA ALT said, "The best way to find out how to engineer solutions is to get out with the guy who uses them." These principles are captured in the components for good business.



Note: [1]Documents are created for critical, high-risk, unclear, complicated tasks, and external interfaces.

Figure 1: *The Defined Process*

Maneuver Control System Light Program

The feasibility and benefits of applying the components for good business can best be illustrated by way of example. The Army's Maneuver Control System (MCS) program is part of the Army Battle Command System and provides the commander with the capability to plan and monitor the battle. The MCS program is managed by the product manager (PdM) MCS, under the program manager Ground Combat Command and Control and Program Executive Officer Command, Control, Communications – Tactical. Development is led by the Air Mobility Command Communications-Electronics Command (CECOM) Software Engineering Center and supported by Shonborn-Becker Systems Inc., L3 Ilex, Lockheed Martin, CECOM Research Development and Engineering Center, and others.

MCS Light was born out of opportunity and necessity. The MCS Light product implements command-and-control functionality on a PC/Notebook/Windows platform. The MCS Light product has gained widespread acceptance within the Army command-and-control user community. Representative of the success of the product are comments made by Lt. Gen. John Vines, previously the commander of the 82nd Coalition Task Force (CTF82) in Afghanistan and currently the commander of the 18th Airborne Corps, who wrote:

> MCS Light is the best tool available today … recommend the Army adopt CTF82's employment of MCS-Light as its strategy to rapidly deploy a standard, interoperable, digital command-and-control system Army-wide. [7]

MCS Light has become the planning tool of choice for nine out of 10 active Army divisions. Much can be learned from examining this success.

The MCS Light development process is the result of several years of direct experience developing software in a very dynamic environment. The process is well defined and has been, in fact, in use for several years. And the process has resulted in a high-quality product that has been widely accepted by the user community. Figure 1 is a graphical representation of the defined process.

One of several key aspects of the process is the notion of iterations and releases. The MCS Light project has adopted a four-week iteration and a three-month release cycle. Within an iteration, a team may cycle through the design, construct, integrate, and demo steps several times. This can be done within a team and also across teams within the project. It is also important to note the level at which the definition of what is in a release and what is in an iteration is managed. At the release level, agreement is reached with the PdM. Definition and modification at the iteration level is managed at the project-leader level. Again, this provides for the flexibility needed to effectively manage in a very dynamic environment

Balance Between Plan-Driven and Agile Development

The MCS Light software process has struck a balance between agile development and plan-driven development, or planned agility in the following ways.

Individuals and Interactions Over Processes and Tools

The MCS Light project and its broader organization have consistently placed sig-

nificant emphasis on individuals and have backed up this emphasis with investment. Roughly half of the development team are government civilian employees, the other half are contractors working on-site as an integral part of the team. Software developers represent more than 85 percent of the project staff, and all civilian engineers have either completed or are pursuing advanced degrees in software engineering.

Consistent with agile development approaches, the overall development team is comprised of smaller teams. These teams typically consist of three to 10 individuals who are co-located within the same office. Interaction is informal, constant, and essential to the approach.

Working Software Over Comprehensive Documentation

The MCS Light project has placed considerable emphasis on the software product and has considered extensive documentation as a significant distraction from developing the end product (more than 800,000 source lines of code). Therefore, it concludes that developing such documentation represents an even greater risk than it is intended to avert.

The architecture is extensively documented and that documentation is maintained. In addition, an "MCS Light For Dummies Guide" has been developed as a training guide for users. Additional training documentation has been and will be developed to an even-greater degree as fielding of the MCS Light product progresses. There is also documentation that traces planned and delivered functionality back to the system Operational Requirements Document.

The product itself, as opposed to extensive documentation, has served as the basis for interactions between the user and the development team. Relatively speaking, little documentation has been developed on the MCS Light project, and no one has missed what has not been developed, including those paying the bills.

Customer Collaboration Over Contract Negotiation

A heavy emphasis has always been placed on collaborating with the user. For the MCS Light development team, there is also another customer: the PdM MCS. Interactions with the PdM are frequent and less formal than the requirements-based contracting approach so often implemented within the Army. The overhead associated with detailed contract negotiation – and renegotiation every time a change is necessary – is overly burdensome to any development effort looking to rapidly respond to a customer's needs. The project has adopted the equivalent of a level-of-effort agreement with the PdM. Within this approach, it can measure progress at the standard milestones and measure earned value.

Responding to Change Over Following a Plan

The Army as an institution is well versed in the development of plans. Fortunately, the Army also recognizes that no plan, even the best plan, survives long in a dynamic environment before needing to be revised. Planning for software development is not significantly different than planning for a battle. The MCS Light effort has consistently placed an emphasis on responding to change. This emphasis gives the team the flexibility to respond effectively to the constant evolving and changing user needs.

"The MCS Light effort has consistently placed an emphasis on responding to change."

Get as Close to the User as Possible

On the MCS Light project, the team has been accused in the past of listening too much to the user and the surrogate user, Training and Doctrine Command System Manager (TSM), as opposed to strictly adhering to requirements definitions and programmatic structures. Doing so has served the project well. As stated earlier, as a developer the closer you are to the user, the more likely you will develop something useful. Simply put, that means having software developers and end users working side by side.

On MCS Light, the project leader, all team leaders, and a significant number of project engineers have spent a significant amount of time in the field with users. MCS Light software engineers have worked side by side with users in garrison, at war-fighting exercises, and have even deployed with units to Afghanistan and Iraq. Being that close is harder than not, but it is the only way to develop a useful product.

Show Them, Ask Them, and Repeat Often

Key to the MCS Light success has been establishing a Beta Site concept. Leveraging industry practices, some operational units were identified as official Beta Sites. As a Beta Site, the units were provided with developmental releases of software. The premise was simple: the team would provide incremental releases of software, the user would provide feedback, and the team would respond rapidly where possible with another incremental release.

Instead of having to wait years for a new version of software that would likely not satisfy their needs, users were rapidly and frequently given developmental releases of software that, incrementally, met more and more of their needs. Confidence and trust between the developers and the users were formed. With trust comes the need for less bureaucracy, thereby enabling the streamlining of the approach even more. Since its inception, every active Army division has come online and requested to become an MCS Light Beta Site.

The benefits of this approach cannot be overstated. Through this approach, it is worth noting that Army units have demonstrated a willingness to accept good enough software much sooner over the promise of better quality software much later. If they do not like the product delivered, or the product delivered does not work, the user has no problem saying so.

The best case is that the team rapidly responded to the user's need and got valuable feedback as to what else was needed. The worst case is that the team learned what the user did not want or need, and only lost the time invested since the previous release. In that respect, the Army is no different than commercial industry – time to market or time to field is a priority, and only an agile approach will do.

Architecture, Architecture, Architecture

It would not have been enough to simply be close to the user and provide early and frequent development releases. The product also needed to be sound and evolvable. From the onset of the project, architecture definition and evolution has been a cornerstone of the development effort. Software architecture was defined almost from day one, and a well-defined architecture has been kept up to date and has served as the basis for all development efforts.

Also key to the success of the project has been a well-structured architecture. In the case of MCS Light, a three-tier architecture was defined and adopted. This

architecture has served the project well in allowing developers to leverage COTS products and tools across the different tiers as well as in providing a powerful approach to managing data. While everyone talks about how important architectures are, MCS Light as a project has actually implemented an architecture-based approach to development, and the continued evolution of the product is the best testimony to that case.

Summary

Insanity has been defined as doing the same thing over and over again and expecting different results. If the Army software community is to truly, that is truly, achieve gains in effectiveness and efficiencies, it must be willing to abandon those practices that have not served it well. The Army must be willing to adopt practices that strike a balance between discipline and agility.◆

References

1. U.S. Air Force Software Technology Support Center. "Agile Software Development." CROSSTALK 15.10 (Oct. 2002) <www.stsc.hill.af.mil/crosstalk/2002/10/index.html>.
2. Paulk, Mark. "Agile Methodologies and Process Discipline." CROSSTALK 15.10 (Oct 2002): 15-18 <www.stsc.hill.af.mil/crosstalk/2002/10/paulk.html>.
3. Boehm, Barry. "Get Ready for Agile Methods, With Care." IEEE Computer Jan. 2002.
4. Agile Alliance. "Agile Software Development Manifesto." 13 Feb. 2001 <www.agilealliance.org>.
5. U.S. Army. "Transforming Current Operations." Association of the United States Army Acquisition Symposium, Falls Church, Va., 8 Sep 2003.
6. Vines, M.G. John. "Commander CTF82, Memorandum Thru Commander CTF180 and Commander U.S. Central Command For U.S. Army Deputy Chief of Staff for Plans and Operations." 15 Jan 2003.

About the Author

John S. Willison is director of Advanced Battlespace Solutions for the U.S. Army Communications Electronics Command (CECOM) Software Engineering Center, Fort Monmouth, N.J. CECOM is responsible for developing software architectures and products for Communications, Command, Control, Computer, Intelligence, Electronic Warfare and Sensors systems. Willison is experienced in the application of software technology, software architecture, prototyping, and management. He has received numerous awards, including the Army's Distinguished Service Award, the Secretary of the Army Award for Outstanding Achievement, the Federal Technology Leadership Award, and the Federal 100 Award. Willison has a Bachelor of Science in electrical engineering from Lafayette College and a Master of Science in software engineering from Monmouth University.

CECOM Software Engineering Center
ATTN: AMSEL-SE-AT
Fort Monmouth, NJ 07703
Phone: (732) 532-2342
E-mail: john.willison@us.army.mil

Applying Decision Analysis to Component Reuse Assessment

Michael S. Russell
The Anteon Corporation

Reusing and combining components of existing systems to create new ones provides a cost effective and flexible method for developing new systems, and is one of the keys behind component-based architecting. However, achieving these benefits in the real world is never easy. The challenge to system developers is not only determining which reusable components to evaluate for possible incorporation into the new system, but also defining what constitutes a reusable component for that system. This article proposes a methodology for applying decision analysis to support component reuse assessment.

New system development has been driven toward component reuse by many factors, including the emergence of rapidly changing technology, faster development timelines and limited budgets, the inability of *stove-piped* legacy systems to deal with information-/network-centric warfare, and the emergence of new system acquisition policies. Reusing components of existing systems is a viable method to overcome these factors. Effective reuse allows system development to stay current with technology, react to tightening schedules and budgets, and manage development risks.

Realizing the benefits of component reuse for the federal government is one of the U.S. Chief Information Officers (CIO) Council's focus areas. The council is currently developing overarching guidance to define component-based architecting for the federal government and the role of component reuse within it [1]. However, the CIO council's objective is not to dictate explicit reuse assessment procedures, rather, it hopes to provide the high-level guidance necessary to set out the scope of such a program.

Similarly, the U.S. Navy is in the midst of developing a component-based shipboard computing environment called the Open Architecture Computing Environment [2]. The program seeks to specify a broad range of reusable components and entire applications upon which to build new systems. In both cases, and for the engineering community at large, a standardized reuse assessment process would be beneficial.

Upon implementing a component reuse assessment process, the first question that comes to the developer's mind is usually: "Which components should I evaluate for reuse potential?" For almost any system development effort, there are an overwhelming number of reusable components that might be considered, and attempts to exploit these components through using traditional engineering approaches have been disappointing [3]. A structured and tailorable decision-making process, incorporating both qualitative and quantitative analysis, is needed to filter out components to be evaluated, select the right reuse candidates, and justify reuse decisions.

This article advances the current state of research in the application of decision-making processes for component reuse by focusing both on the actual evaluation processes and the filtering mechanisms that must be in place to support a successful process application. Most programs have an extremely large group of potentially reusable hardware and software components from which to choose. Being able to objectively filter this group and develop a shorter list of the most likely reuse candidates for in-depth evaluation is critical to meeting budget and schedule constraints.

> *"... the assessment process is designed to be used as a decision aid to the development team, not to dictate the decision."*

Methodology

The methodology expands upon previous work sponsored by the Software Productivity Consortium (SPC) [4], the Institute of Electrical and Electronic Engineers (IEEE) [5], and others [3, 6, 7]. The SPC's Comparative Evaluation Process (CEP) is a method for quantitatively evaluating software reuse components. IEEE Standard 1517 defines reuse evaluation considerations as part of a software engineering life cycle, and lays out a method for establishing a software code reuse library.

The methodology uses aspects of the CEP and life-cycle implementation guidance from the IEEE. It adds qualitative assessments as an initial reuse candidate filter, extends the method to add hardware component assessment, and defines assessment criteria categories covering functional, technical, and programmatic evaluation issues.

The methodology is comprised of the following steps: bounding the evaluation effort, identification of candidate components, defining evaluation criteria, evaluating alternatives, and analyzing assessment results.

Bounding the Evaluation Effort

The first step is to define the scope of the reuse effort. This presupposes the creation of an operational concept, requirements specification, architecture, or other statement of expected system functionality, interfaces, and deployment concept. These specifications form the framework upon which reuse decisions are built. In the specification, the developers will have allocated desired system functional requirements to objective or conceptual hardware and software system components. This allows the assessment team to generate a list of candidate components during the next step.

During a typical system life cycle, the system specification will normally stop shy of identifying design-level details or mandated implementations that overly constrain the developers design space, while including enough detail to ensure compliance with desired technical standards, legacy interfaces, and other constraints. However, heavy reuse of existing components mandates some changes to this approach. The specification for a system featuring reuse will have many of its design elements predetermined from the start.

These constraints should be identified early in specification development if known. Most likely, some reusable components will be known at this time and will influence specification development. However, it is just as likely the developers

will not know or only have a general idea about what types of reusable components they want to incorporate into the system. This mandates a system life-cycle model that allows the specification to be modified after the best reuse component candidates have been identified, assessed, and selected for incorporation into the system. Additionally, the integration of many reusable components may bring up compatibility and interface problems that were not readily apparent even after several system design iterations. So a program manager desiring to use any level of reuse must be willing to revisit the original specifications as needed to ensure the resulting system meets the customer's requirements.

Identification of Candidate Components

Individual system requirements, allocated through the systems architecture, could potentially be met with different combinations of hardware, software, and business processes. Programmatic requirements to maximize using reusable components complicates system design by adding additional variables such as proprietary protocols, hidden or inaccessible algorithms, and emergent functionality that appears after the system is wired together.

Additionally, developers have to determine which portions of the system are best built using reusable components, which should be based on custom development and how business processes are supported by each. Together, potential combinations of process and technology form a multi-variable quandary for system developers.

Identification of candidate-reusable components should start by deciding what required functionality should be instantiated by reusable components. The functionality to be replicated forms the basis for the initial selection of candidate reuse components. Second, programmatic or legal requirements such as a mandate to use *component X* for all new systems development, will flesh out the initial list.

The initial list of candidates will take some research and may result in an extremely large set of components. In some cases the desired functionality may be replicated using reusable hardware only, reusable software only, or some combination of both. At this point in the process, the developers should not filter out potential reusable components on an ad-hoc basis; rather, developers should seek to identify as many potential component options as possible to support a variety of systems design options and objective decision-making.

Category	Definition
Functional Requirements	Those requirements that outline the expected behavior of the system. This behavior is required for the system to perform its intended purpose.
Non-Functional Requirements	Those requirements that address expected behavior or other aspects of the system that are not required for the system to perform its intended purpose but serve as an enabler.
Technical Requirements	Requirements that specifically define technical constraints and interfaces that the components must adhere to.
Programmatic Requirements	Requirements arising from budgetary, schedule, or resource constraints. For instance, a component that will not be available until after the system is delivered would not meet a schedule requirement.

Table 1: *Evaluation Criteria Categories*

Defining Evaluation Criteria

There are four categories of criteria that will be used to assess each reuse candidate. Due to the variety of reuse candidates being assessed, some candidates may not receive an evaluation in each category; however, this will be the exception. Furthermore, these categories serve as the starting point. Since each project is different, additional categories may need to be defined to evaluate required component functionality – or functionality that the component should not exhibit. Table 1 defines these categories.

> *"The determination of which evaluation category takes precedence will be different for each system ..."*

Functional and non-functional requirement criteria are derived from the system's specification document or other document that outlines the system's requirements. If the criteria were derived from a functional decomposition, assessments will be generated for the lowest level of decomposition (the leaf node level). Ensure the matrix contains the entire functional decomposition; otherwise, dependencies relationships between functions will be missing, and the assessment results may become skewed.

Technical and programmatic category criteria are derived from industry standards, government policies, best practices, and other documents. This set of criteria tends to focus more on the hardware and physical interoperability of the reuse candidates. For reuse candidates that can be separated into hardware and software components, separate assessments will be conducted. Conversely, reuse candidates that cannot be separated into hardware and software components will be assessed as one system. The reusability criteria in these sections generally try to answer the following types of questions:

- Are the reuse candidates compatible with existing government and industry technology standards and mandates?
- Are the reuse candidates compatible with existing organizational standards, processes, and other organizational-specific mandates?
- Are the reuse candidates mature and stable enough to ensure their long-term viability as a part of the system's design?
- Are the reuse candidates sufficiently documented to allow rigorous analysis of their functionality, interfaces, and potential for integration with other components?
- Have all schedule and budget issues associated with the reuse candidates been considered and documented?

Next, each evaluation criteria is assigned a weight. This weight is used to judge the relative importance of a specific criterion to other criteria regardless of category. For example, if the system has five functional requirements, only four may be deemed critical giving them a weight of 0.9. Table 2 contains the criteria weights and definitions.

A system specification typically addresses many requirements that are *nice to have* versus essential for system success. For instance, the ability of a word processor program to check the spelling of a document may be critical (a score of 0.9), while the ability to check document spelling in real time while the user types

Table 2: *Criteria Weight Definition*

Weight	Definition
0.1	Failure to meet this criterion is minimal, or its impact on the system can be mitigated.
0.5	This criterion is an important contributor to system capability, but not essential.
0.9	This criterion is critical to system success

Component Name: (My Component)			Reviewer: (Name)
Criteria Number	Functional Requirements Category	Assessment Result	Reviewer Notes
1	Component Cost		
2	Size of User Community		
3	Maturity		
4	Integration Cost		
5	Schedule Delay Imposed by Using That Component		

Figure 1: *Initial Assessment Matrix Example*

the document may not be critical (a core of 0.5). The correct weighting for each criterion may be derived from many sources, including:

- Customer requirements and expectations.
- System requirements volatility.
- Technical maturity of the system domain.
- Statements of *must have features versus nice-to-have features*.

Once the criteria have been defined and weighted, evaluation matrixes listing each criterion will be generated. The purpose of the matrix is to standardize the evaluation process, thus providing a clearer and more defendable reuse evaluation result and component recommendation. Example matrixes are discussed in the next section.

Evaluating Alternatives

Initial Assessment: Qualitative

The initial assessment is qualitative and used as a mechanism to filter the set of reuse candidates to select the two or three candidates that offer the best match to the system's requirements. The actual number selected will depend on the amount of resources the program can expend to support more detailed assessments. To conduct the initial assessment, the evaluator must first identify the modules (subcomponents, software segments) of the reuse candidate. The individual modules, rather than the component as a whole, should be assessed against the criteria. The objective is to understand which portion of the reuse candidate fulfills the criteria.

The evaluator will fill out a separate criteria assessment matrix for each reuse candidate. For instance if there are four candidates, the evaluator should have four completed matrixes at the end of the assessment. The assessment matrixes will be used to help the system developers filter though the complete set of reuse candidates, thus providing them with a rationale for choosing the most likely reuse candidates for further analysis. Figure 1 contains an example initial assessment matrix.

To record the assessment, a *1* is put into the *Assessment Result* column if the reuse candidate fulfills the criteria; a *0* is inserted if not. In general, the criteria should be rather loosely interpreted, as this qualitative assessment is designed to determine only if the reuse candidate should be assessed in more detail later on.

After the evaluation team has finished the initial assessments on each candidate component, the system's stakeholders will use this evaluation as a decisional aid to select the most likely reuse candidates. The selected reuse candidates will undergo a more thorough evaluation during the detailed, quantitative assessment.

Table 3: *Assessment Methods*

Assessment Method	Weight	Definition
Verified	1.0	Verified by the evaluator using hands-on examination in a lab environment.
Demonstrated	0.7	Witnessed by the evaluator in a focused demonstration by an experienced user.
Observed	0.5	Seen by the evaluator but not studied in any depth.
Reported	0.3	Described by a user, associate, or vendor, or seen in vendor or third-party literature.

Table 4: *Reuse Candidate Scoring*

Score	Definition
0.1	No capability to meet the criteria demonstrated.
0.3	Meets <50% of requirements and/or customization not possible.
0.5	Meets 50% of requirements and customization is possible.
0.8	Meets 90% of requirements and customization is possible.
0.9	Meets all system requirements.
1.0	Exceeds system requirements and allows further growth opportunities.

Detailed Assessment: Quantitative

The detailed assessment is quantitative in nature and is used to select the optimal reuse candidate to fulfill each system requirement. Each reuse candidate selected for further assessment during the initial analysis will be evaluated again using one of the methods in Table 3. The selected assessment method will have a significant bearing on the overall evaluation of that reuse candidate, and becomes one of the variables that figures into the component's overall evaluation. For instance, a hands-on evaluation of a candidate component will render better information than simply reviewing marketing literature on the same component.

For large reuse candidates, candidates with many subsystems, or candidates that encompass a mixture of hardware and software, it is conceivable that multiple assessment methods may be justified. Another situation in which multiple assessment methods might be used is when the reuse assessment must be conducted under a compressed timeline. In this case, the evaluators may decide to conduct hands-on testing on the most critical functionality that the reuse candidate must exhibit.

To perform the detailed assessment, the evaluator will determine a score based on the assessment results that indicate the extent to which the reuse candidate fulfills the requirements and metrics of each system criterion. The scores and their definitions are included in Table 4.

This assessment will result in two scores: a criteria score and a composite score. The criteria score is a weighted assessment of the reuse candidate's ability to meet the requirements of each individual criterion. The composite score is the overall score for each category. The process for deriving the criteria score is as follows:

criteria score =
(criteria score) x (criteria weight) x
(assessment method)

The composite score is derived in this way:

$$\text{composite score} \sum_{i} = (\text{criteria score}_{i})$$

where:

$$\text{criteria score}_i = (\text{criteria score}_1 \quad \text{criteria score}_n)$$

The component's detailed assessment can be captured in a matrix similar to Figure 2.

Analyzing Assessment Results

The final step in the process is to analyze the assessment results and select the reusable components that become part of the new system. At this point, the system developers will have to make a critical determination. What is more important: integration flexibility, functionality, programmatic mandate adherence, or some other concern?

For instance, a candidate that implements a required function, but faces integration challenges, may be chosen over a competing candidate that does not implement the function as well but is easier to integrate into the overall system. The determination of which evaluation category takes precedence will be different for each system, and will result in category-specific weights – i.e., functionality is twice as important as programmatic mandate adherence.

The candidate score represents the summation of the functional, non-functional, technical, and programmatic categories with program-specific category weighting. It is used along with the other component scores as an aid to the system developers so they can decide which reuse candidates to incorporate into the system. The procedure used to calculate the scores is the following:

$$\text{candidate score} = \sum_i (\text{composite score}_i) \times (\text{category weight})$$

where:

$$\text{composite score}_i = (\text{functional composite score, non-functional composite score, technical composite score, programmatic composite score})$$

At this point, determining the best reuse candidate seems as simple as selecting the reuse candidate with the highest candidate score; however, this can be misleading. The process is only designed to guide the selection of components by mathematically assessing those components and providing a means by which a reasonable comparison between components can be made. As such, the assessment process is designed to be used as a decisional aid to the development team, not to dictate the decision. The benefit of the process to the decision maker is a structured, repeatable, and defendable process on which to base system design decisions.

Component Name: (My Component)			Reviewer: (Name)		Component Source:	
Criteria Number	Functional Requirements Category	Quantitative Assessment	Criteria Weight	Weighted Assessment	Assessment Method	Criteria Score
1	Component Cost					
2	Size of User Community					
3	Maturity					
4	Integration Cost					
5	Schedule Delay Imposed by Using That Component					
				Composite Score		

Figure 2: *Evaluation Matrix Example*

Conclusion

The Missile Defense Agency's National Team for Command, Control, and Battle Management successfully implemented this approach as part of their block 2004 systems development [8]. This project demonstrated that the ability to objectively judge the suitability of individual components is a critical part of the design process. Achieving the full benefits of component-based architecting depends upon making objective and optimal reuse decisions. System developers must implement decision analysis into their component reuse assessment process, and use this assessment to guide their component reuse decision-making process.◆

References

1. Chief Information Officers Council. Component-Based Architectures and the Federal Enterprise Architecture; draft v1.6. Washington, D.C.: Federal Enterprise Architecture Components Subcommittee, 2003.
2. Program Executive Office, Integrated Warfare Systems. Open Architecture Computing Environment Design Guidance, v1 (Interim). Washington, D.C.: U.S. Navy, 2003.
3. Albert, C., and L. Brownsword. "Evolutionary Process for Integrating COTS-Based Systems: An Overview." Technical Report CMU/SEI-2002-TR-009. Pittsburgh, PA: Software Engineering Institute, July 2002.
4. Phillips, B., and S. Polen. "Add Decision Analysis to Your COTS Selection Process." CROSSTALK April 2002 <www.stsc.hill.af.mil/crosstalk/2002/04/phillips.html>.
5. Institute of Electrical and Electronics Engineers. IEEE 1517-1999: IEEE Standard for Information Technology – Software Life Cycle Processes-Reuse Processes. New York: IEEE Standards Board, Jun. 1999.
6. Kang, K., et al. "A Reuse-Based Software Development Methodology." Special Report CMU/SEI-92-SR-4. Pittsburgh, PA: Software Engineering Institute, Jan. 1992.
7. Griss, M. Architecting for Large-Scale Systematic Component Reuse. Palo Alto, CA: Hewlett-Packard Company Laboratories, 2000.
8. National Team for Command and Control, Battle Management, and Communications. NTB Program Directive: Component Reuse Assessment Process. Washington, D.C.: Missile Defense Agency, Nov. 2002.

About the Author

Michael S. Russell is chief architect and technical director for the Anteon Corporation's Center for Missile Defense. He currently supports the U.S. Navy's Open Architecture Program, and has served as lead architect on numerous federal, Department of Defense, and industry enterprise architecture efforts. He is a visiting lecturer with the Federal Enterprise Architecture Certification Institute, and is a member of the International Council on Systems Engineering's central Virginia chapter. Russell has taught courses in Systems Engineering and Architecture Development for the past seven years. He has a master's degree in system engineering from George Mason University.

The Anteon Corporation
2231 Crystal DR
STE 600
Arlington, VA 22202
Phone: (703) 864-1258
Fax: (703) 521-1027
E-mail: mrussell@anteon.com

Better Communication Through Better Requirements

Michael J. Hillelsohn
Software Performance Systems

Everyone involved with a software development effort needs to have the same understanding of the meaning of the requirements for the application under development. This article describes several techniques that can be used during analysis to assure that all stakeholders reach a common level of understanding.

What makes a requirement effective? The question hangs in the air of the requirements class I am teaching. This is how I start the class, with this simple question. Participants eventually, cautiously give answers that cover the usual array of what makes for a good requirement: unambiguous, testable, clear statement of need, measurable, functionally worded, etc. All correct, but rarely do I get the answer I am really looking for: An effective requirement communicates clearly to all parties who read/ hear it what a piece of software needs to do to satisfy the user's expectations. When your goal for well-written requirements is to communicate clearly, then you can use some techniques to enhance the communicability of the requirements.

The entire requirements definition process is, intrinsically, an ongoing communications process. The customer states the problem that needs to be solved, then explains the job that system users will perform when solving the problem. The developer translates these statements into functional requirements then asks the customer, "Did I understand you correctly?"

Through a series of successive approximations, the developer's documentation of the required system functionality closely approximates the customer's expectation of operational system performance. Yet in some instances, the communication process breaks down. By applying some simple, very specific techniques, the risk of misinterpretation is effectively reduced along with the resulting expensive rework later during the project's development life cycle. Here are four techniques that Software Performance Systems uses to foster communication during the requirements definition process:

- Train the requirements analysis team on project standards.
- Build a requirements reserved word list.
- Clearly define quality requirements with the customer.
- Conduct requirements *scrubs*.

Used by all parties in the requirements definition process, and regardless of the form of requirements documentation, these techniques enhance communication throughout the project and assure quality output.

Train the Requirements Team on Project Standards

Project standards for requirements include processes and procedures for requirements management activities as well as formats and templates for outputs. Using an event-driven learning (EDL) approach, short training experiences are interjected in the process immediately prior to executing a process step. The training experiences focus on performing the next step in the requirements elicitation, documentation, and management process so that everyone is *on the same page* when the process activity takes place. It is particularly helpful if both the customer and developer participate in these training experiences. That way, the customer has an opportunity to influence the way the activity is carried out in his or her organization.

The process for developing and conducting these training sessions starts with the selection of requirements' formats. Involving the customer in template selection communicates what to expect as a final product of the requirements definition effort. As usual with process definition, looking at the output of the requirements definition process provides an indication of what the templates need to contain. For example, if the output is going into a requirements management tool then a decision needs to be made whether to let the tool number the requirements or whether to assign numbers to each requirement regardless of the tool's numbering strategy.

In general, the templates will come either from the artifact templates of the selected software development life cycle or an industry standard such as IEEE 830-1998. In either case, there is a likelihood the template will be tailored to suit the project needs such as defining additional attributes (i.e., priority, source, status, planned release, etc.) for the requirements. Once the templates are defined, then EDL is used to communicate the formats and contents to all participants. With all stakeholders on the project working from a common set of templates, everyone knows what to expect, in terms of artifacts, and is familiar with the contents of each section.

A variety of training experiences need to be tailored to the expected artifacts from the requirements management process. If a requirements management plan is being prepared, the training may consist of little more than reviewing the template and discussing sections of the document with the authors before it is written. If the interim artifact is going to be a use-case or software requirements specification (SRS), then more extensive training experiences are warranted.

The classes will have much similar content such as properly formatting the requirements statement as a single action without using any conjunctions, except in the case of Boolean logic. Instruction is tailored to conventions used for each of the different artifacts. For example, to easily interpret the flows in a use case the labeling conventions are defined to uniquely identify each action in the use case. Thus the use-case writer encloses alternate flow identifiers in parentheses next to the primary flow statement. Figure 1 shows some of the conventions adopted for writing use cases, and taught in the Defining Requirements with Use Cases class.

Adopting such conventions — and ensuring that everyone on the project knows how they are used and what they mean — allows the project to develop its own notation while ensuring that everyone can read and unambiguously understand the artifacts. During training, the requirements analysts have an opportunity to apply these conventions as they practice writing use cases immediately before they write the actual ones. SRS authors, on the other hand, spend more of their training time practicing writing and decomposing properly prepared, testable *shall* statements.

Regardless of the class, the emphasis is on writing requirements that clearly convey to users, testers, and designers what is to happen on the system. The total quality management concept of preparing output that is usable by internal and external customers is especially valid when training people to write better requirements. All potential requirements' *customers* are considered as the requirements are written and reviewed.

Build a Requirements Reserved Word List

Words and terminology are the primary communication among people. Word-based communication can be imprecise because the same word heard/read by different people can have different meanings, different words can mean the same activity or thing, and different groups of people have their own jargon. Unified Modeling Language attempts to address these issues with heavy use of diagrams with well-defined components; however, this approach does not help with textually expressed requirements.

The inclusion of a glossary as an artifact in the Rational Unified Process defines domain-specific terminology, but does not provide for unambiguous expression of what the system shall do when it is built. Some early programming languages either specified or allowed the programmer to define a set of reserved words for the program. They allowed the programmer to communicate unambiguously with the computer to perform specific operations consistently. This concept transfers effectively to requirements engineering.

A reserved word list for requirements contains both general and domain-specific terms. General terms select a single word to describe a specific operation. For example, you can specify that the system needs to append information to a database by saying that it shall *save*, *record*, *capture*, or *store* a data element. A team of requirements analysts may use any or all of these words in their requirements; when the tester, designer, or customer reads the requirements, they may or may not interpret each word differently with different tests, designs, or acceptance criteria for each requirement. It would be clearer to define *record* as the only verb to be used to update a database that maintains historical information for the system. A clearly bounded definition like this also makes the other verbs available for things like temporary and local information storage by the user.

There are no synonyms in the reserved word list because they compromise precision of expression. In case-management operations, for example, the terms *case*, *file*, and *work-item* are often used interchangeably. Sometimes in the user community, these terms are used differently in different parts of the organization. It is critical that a single term is used to define the object that is being manipulated by the system so that everyone knows what the requirement is acting upon. In this instance, discussions with the user community will arrive on a consensus term and its definition. Occasionally the discussions with the user may result in nuanced definitions where multiple terms are used but each has a unique attribute. For example, *case* may be used to refer to the offline version while *file* refers to the online version of the same information.

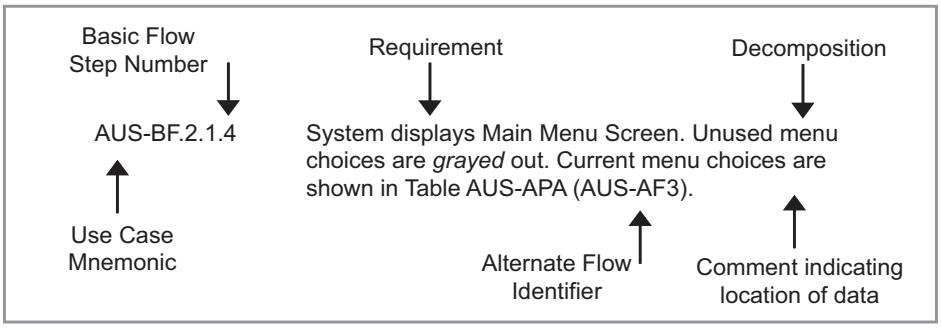


Figure 1: *Defining Standard Notations for Use Cases*

Most words in the reserved word list are verbs with very specific meanings. Table 1 shows three specific terms for user/system interactions (from the user/actor perspective) that clearly communicate the nature of the interaction. The fourth allows for the interaction to be more clearly defined during design. Thus when the interaction is known during requirements definition, it is documented within the requirement/use-case so that it can be confirmed during requirements review cycles. If the interaction is not predetermined, a method of documenting the requirement is available that does not preclude design options.

Adverbs such as *automatically* and *may* are also useful additions to the list; they can be shorthand for describing some common constructs in applications systems. For example, *automatically* can be used before a system action when there is no intervening or triggering action by the user; *may* can indicate that the user action described in the requirement is optional. Constructs such as optional user actions and automatic system actions are easily captured and communicated by designating specific reserved words to represent the ideas. If reserved words are not designated or defined, then the representation of the constructs can become convoluted in requirements documentation.

Occasionally, an analyst may use a word that is not on the reserved word list as a system or user/actor action. This is usually detected during the quality assurance (QA) review of the requirements artifact. In that instance, the reviewer should give the analyst the option of either using a word from the reserved word list, or if there is no term that adequately and accurately describes what is going on, then the analyst is asked to add the term and its definition to the list. Over time, the list will grow to meet the exact needs of the application domain. In general, the reserved word list does not grow too large because systems can only do a limited number of things (e.g., print, display, record, calculate, verify, etc.). The reserved word list is included as an appendix with the documented requirements in the SRS or with the supplementary requirements.

Define Quality Requirements With the Customer

The statement "I know quality when I see it" is not a viable means of defining the quality of software applications. Quality requirements must be defined in the beginning of a project so that they are considerations in the design of the architecture and application features. Software quality factors were defined by the Department of Defense to be used in the acquisition process.

Software Performance Systems uses quality factors as a means for customers and

Table 1: *Using Reserved Words*

Reserved Word	Definition
Enter	A situation where there is a non-specified form for the user to provide information to the system. *Enter* is used where the specific entry method (typing, indicating) is either to be defined during design or when there is more than one way to provide the information. In the latter case, the comment field should contain the acceptable alternate forms by which the user can provide the information.
Indicate	The user makes a binary choice by setting an online Yes/No, On/Off, or other type of flag (e.g., by marking or inserting a symbol such as *comma* [,] in a check box), or otherwise select a setting from an online indicator. A single mouse click or screen touch is a typical method of interaction.
Input	Keyboard entry of information by a user (e.g., typing) requiring an end-of-input signal (i.e., [Enter]).
Select	Choose from a set of options displayed on a list.

Quality Factor	Definition	Components
Efficiency	Relative extent to which computer resources are utilized.	Communication, Processing, Storage
Integrity	Extent to which security protocols are implemented to guard against unauthorized use and access to software and data.	Access Control, Virus Prevention
Reliability	Extent to which the software can perform without any failures.	Accuracy, Anomaly Management, Simplicity
Usability	Relative effort required by the user community to use the application.	Operability, Training, User Support
Correctness	Extent to which the software conforms to its standards and specifications.	Completeness, Traceability, Accuracy, Consistency
Maintainability	Ease of effort associated with finding and fixing a software failure.	Consistency, Visibility, Modularity, Simplicity, Documentation
Verifiability	Relative effort required to test the software operation and performance.	Visibility, Traceability, Documentation, Simplicity
Expandability	Relative effort to increase the application's capabilities and functionality.	Functional Scope, Virtuality, Modularity, Documentation
Interoperability	Ability of the software to function on a variety of platforms and operating systems.	Commonality, Common Functions, Independence, System Compatibility
Portability	Relative effort required to transport the system to another environment.	Independence, Modularity, Documentation
Reusability	Relative ease of using software components, unchanged in other applications.	Application Independence, Clarity, Document Accessibility, Modularity

Table 2: *Selecting Quality Requirements During Analysis*

end users to express how a system satisfies their highest priority, non-functional needs when it is in production. The basis for defining what quality really means to the customer is to have the customer select the quality factors that are most critical to the effort, and then define requirements that set clear, meaningful, attainable, and measurable quality attributes for the application. When these requirements are met, the customer will have a quality product as defined in their own terms for this specific application.

The 11 quality factors shown in Table 2 are derived from the work sponsored by Rome Air Development Center (performed by Boeing Aerospace) in the mid-80s. The user is asked to prioritize the factors since the application cannot consider all of them as the most important. Prioritization involves discussing each of the factors among the group, then each group member selects their three most important factors.

Customers may balk at having to select just a few factors, but the process of prioritization makes them focus on what is critical for the system's success in their environment. For example, although *correctness* is always important, it may not be as critical for a case-management system as it is for a financial system. Likewise *usability* is a higher priority when there will be a large user base than when there are only a few trained users.

After the votes are tallied and presented to the group, another discussion ensues to arrive at a consensus of the three most important quality factors. These discussions may be the first time that members of the customer/user community really try to articulate how quality is defined for their application. While the development team certainly does not ignore factors that are not the highest priority, their importance is secondary; they are not decomposed and measured the same way as the high priority quality factors. By going through the selection and prioritization process, the user community communicates among themselves and to the developer those factors that are most critical and a little less important in the final product.

Quality requirements now need to be defined so everyone agrees on observable criteria for determining if the high-priority quality factors have been achieved. This step requires the user to translate the abstract definition of the quality factor into operationally meaningful terms. When defining reliability, the facilitator (F)/user (U) interaction typically goes something like this:

F: You decided that you need a reliable system. How reliable does it have to be?
U: Oh, very reliable.
F: Can it go down for an hour once a year?
U: Sure, no problem.
F: How about if it goes down once a month? Does that endanger you accomplishing your mission?
U: I suppose once a month may be acceptable, but it depends when.
F: What activity is so important that it absolutely cannot go down during the activity?
U: When agents are uploading case files.
F: Okay, so with the exception of the time when the agents are uploading case files, would it be acceptable if it goes down once a week?
U: That would be the absolute most that we could tolerate, but only if it was down for a very short period of time. And so on.

Now requirements define when the system cannot have a failure and that the mean time between failures is at least 40 operational hours. Continued discussion defines the acceptable time limits on repair/down time under various circumstances and other aspects of system reliability. Shown below are samples of requirements that a specific user group arrived at when defining reliability for their system.

- **QFR.1:** Each iteration of the system shall be verified before it is put into production.
- **QFR.1.1** Each iteration of the system shall have no critical system failures after it is placed in production.
- **QFR.1.2** Each system iteration shall have one or less major defects arise during the next six months in production.

All of these requirements are observable and measurable. Some can be verified during testing, but most must wait until the system is in production to verify that they have been satisfied. They clearly communicate to all parties what the user means by saying the quality system is reliable. The developer uses these requirements during design to specify a system that is most likely to meet these quality requirements (e.g., degree of component architecture, redundancy, etc.). If these and other quality requirements are not communicated to the development team early in the development life cycle, then the probability that the delivered system meets the users' quality expectations is reduced.

The biggest benefit of defining quality requirements in such specific terms during analysis is to communicate these requirements within the user community. It is common that the developer's client and the end-user of the system are different parts of the sponsoring organization. It is critical for system success that the entire sponsoring organization agrees on what constitutes a quality delivered system. Reducing ambiguity in defining quality among all system stakeholders leads to better assurance of achieving customer satisfaction when the system goes into production.

Conduct Requirements Scrubs

Inspections and walkthroughs are proven techniques for early defect detection and are considered to have a significant positive impact on the *cost of quality* of software products. We should do formal inspections on requirements artifacts, but often the potential participants are put off by the formality

of the process. So Software Performance Systems introduced the term scrubbing the requirements, which is less formal but achieves the same end as formal inspections.

During a thorough requirements reading in a group setting, many defects are detected and corrected. In addition, Software Performance Systems often has a navigational prototype available to the participants to help them visualize how the words translate into an actual application.

Where formal inspections rely on a small focused cadre of reviewers (generally set at three to five people), it is imperative that multiple perspectives are represented at requirements scrubs. People present at the scrub should include the following:

- **Producer.** The person(s) who actually wrote the requirements. During the scrub, the producer will read each requirement aloud.
- **Customer.** The requirements are a description of what the customer will receive when the application is delivered, so the customer needs to be present to hear and provide input to the detailed application description. The customer is also the referee when requirements changes are suggested (generally by the user) that may be out of scope.
- **End-User.** The specifics of what can be done with the application are represented by the functionality described in the requirements, so the scrub gives users the opportunity to correct misconceptions about how they do their job. Just a few knowledgeable user representatives who can make decisions for the functional area being reviewed will be productive. Too many users at a scrub leads to long discussions without resolution.
- **Tester.** The tester will determine whether there is enough information present in the requirements to develop test cases to verify that the requirement is adequately satisfied when the application is delivered. It is important that the tester ask for clarification in the presence of the customer and user so that a common understanding about the acceptance criteria for each requirement is established.
- **Quality Assurance.** The scrub is an early opportunity for QA to correct defects related to noncompliance with standards for writing well-crafted requirements. QA can also facilitate the scrub.
- **Technical Lead.** As the user discusses how the requirements are satisfied on the job and the analyst describes the requirements, it is helpful to include technical representation at the scrub to get a feel for how the requirements translate into the real world of the user, and to determine the feasibility of implementing the requirements as stated.
- **Project Manager**. Since the user and/or customer may raise concerns about functionality that may either be in or out of scope for the effort, the project manager may want to attend the scrub. Alternatively, any questions regarding scope may be deferred until consultation with the project manager is possible. In most cases this is preferable rather than discussing scope issues during the scrub.

The scrub is conducted very similarly to a formal inspection. The requirements are distributed to the participants in advance, and they are encouraged to review them and be prepared with comments. In reality, tasking customers and users – key participants in the scrub – to do the review prior to the scrub is problematic. Preparation will make the scrub more efficient, but unlike an inspection it is not cancelled if the participants do not prepare adequately.

At the scrub, each requirement or step in a use case is read aloud and comments are provided immediately after the reading. For a use case, alternative flows are read at the point where they would diverge from the main flow. The comments are recorded either by the producer and QA or a scribe. At this point, the facilitator's task is to keep the discussion focused and brief. It is important to maintain a good tempo at the scrub and not allow it to get bogged down in long discussions on a single requirement.

When the navigational prototype is used at the scrub, the facilitator should steer the discussion and comments away from design issues (how the screen looks) and maintain the focus on requirements and functionality. In general, each session should last for no more than two hours; otherwise, it is hard to stay focused. If more time is needed, break the scrub into multiple sessions. The ideal outcome of each discussion is a reworded requirement that everyone agrees is accurate, verifiable, and feasible. If that end-state cannot be reached quickly during the scrub, defer the discussion for offline resolution.

The dynamics of the scrub foster communication among all the stakeholders in the system. As participants attend multiple scrubs, they get a good appreciation for the overall functionality of the planned application and how the pieces fit together. Like an inspection, scrubs serve an important role in fostering a common understanding and knowledge base about the application early in the product's life cycle. This way misunderstandings can be clarified and issues related to functionality shared among stakeholders, and resolved.

Scrubs also are excellent training vehicles for improving the requirements analyst's knowledge and skills relative to writing well-constructed requirements. At one recent scrub, right after the analyst read the requirement aloud, before anyone could comment, she covered her face with her hands and said, "That was a terrible requirement, I'll fix it." Everyone laughed and the scrub continued. The most recent SRS submitted by this author-analyst, to QA, had only .02 defects per page compared with the .80 defects per page before Software Performance Systems implemented these techniques in 1999.

Conclusion

Generally, the systems built by Software Performance Systems are used by people when they perform their jobs or by the public when they provide input to an agency. If the delivered system is not correct, then the agency is crippled in achieving its mission. That is why Software Performance Systems has focused on the importance of using requirements to assure that communications between the developer and the customer are an accurate reflection of functionality that is required when a system goes into production. Using multiple techniques to enhance the communications process during the inception of a software development effort ensures that the delivered application meets the user's quality expectations and is fit for use by the intended audience.◆

About the Author

Michael J. Hillelsohn is a director of Product Assurance at Software Performance Systems in Arlington, Va. He is a certified quality professional with more than 30 years of experience doing development, management, and performance improvement in software and systems development environments. His multi-disciplinary approach combines quality systems and training expertise to improve the performance of organizations and individuals. Hillelsohn's process-oriented performance-engineering methods facilitate adoption of external frameworks (Capability Maturity Model®, ISO, Baldrige) to improve the quality of organizational products and services.

3141 Fairview Park DR
STE 850
Falls Church, VA 22042
Phone: (703) 839-4055
E-mail: mhillelsohn@goSPS.com
hillelsohn@erols.com

Enterprise DoD Architecture Framework and the Motivational View

D.B. Robi
Lockheed Martin Integrated Systems and Solutions

There is a growing movement toward developing enterprise architectures within the Department of Defense (DoD) and other federal arenas. This typically takes the form of documenting the as-is state, defining the to-be state, and developing a transition plan. To use the DoD architecture framework (DoDAF), the models appropriate to enterprise architecture need to be identified and the shortcomings in business and financial considerations need to be addressed. This article describes using the DoDAF, including the addition of the motivational view to address the shortcomings to accomplish a complete description of enterprise architecture.

In performing any modernization task, there are typically four questions that must be answered: "Where am I now; what is my current *as-is* architecture?" "Where do I need to be; what is my target *to-be* architecture?" "What is the gap or differences between the two?" and "How do I get to the to-be state; what is the transition plan?"

This article describes a hybrid approach to answering these questions by using a subset of the existing Department of Defense (DoD) Architecture Framework (DoDAF) [1] views and adding an additional view to capture all the important and missing business, financial, and technical analysis information. This extension to the DoDAF is defined as the Motivational View (MV).

The Enterprise and Enterprise Architecture

An enterprise's competitive edge and ultimate success are enabled by its ability to rapidly respond to changing business strategies, governances, and technologies. The DoD environment spells this competitive edge as *victory*. The competitive edge translates into higher levels of customer satisfaction, shorter work cycles, and reductions in schedules, maintenance costs, and development time, all resulting in lower overall costs of ownership.

Enterprise architecture is the key facilitating ingredient providing a holistic view and a mechanism for enabling the design and development as well as the communication and understanding of the enterprise. The overarching goals of enterprise architecture are to manage the complexity of the enterprise, align business strategies and implementations, and facilitate rapid change in order to maintain business and technical advantages.

Lockheed Martin's view of an enterprise is a collection of business systems that control and manage the enterprise's functional areas. Enterprise architecture describes these systems in terms of their behaviors, methods of communications, and constraints.

Enterprise architecture enables the high-level prospective and views needed to transform as-is legacy systems of disparate stovepipe applications into the to-be set of modernized, agile, and integrated business processes. Lockheed Martin starts its enterprise architecture documentation by using a subset of the existing DoDAF views.

DoDAF Enterprise Architecture

Enterprise architecture is documented as an organized collection of information in three divisions: driving strategies, baseline, and transition plan. The driving strategies depict goals and objectives and show the way forward, indicating where the enterprise needs to go based on business drivers, policies, rules and regulations, and advancing technology. This provides the to-be target model. The baseline is the current, as-is enterprise architecture documented in graphical models and text describing the current position in terms of organizations, business processes, information, applications, and technologies. The transition plan is the set of initiatives set to a timeline to sustain and maintain the enterprise architecture as vital to accomplishing the strategic missions of the enterprise and transition from the as-is state to the to-be state.

The DoDAF describes a set of 26 work products to ensure uniformity and standardization in the documentation and communication of architecture. A previous version of the DoDAF divided this list into two major categories: essential and supporting. Essential products constitute the minimal set of artifacts required; supporting products constitute information that may be needed depending on the specific drivers of the architecture.

The list of products is further refined into four views: all views (AV) and three architectural views that include operational view (OV), system view (SV), and technical standards view (TV). Briefly characterized, the AV is the overarching information describing the architecture plans, scope, and definitions. The OV focuses on the behaviors and functions describing the DoD mission aspects, both warfighting and business. The SV describes the systems and applications supporting the mission functions. The TV describes the policies, standards and constraints. The current DoDAF version indicates a subset of work products that should be developed at a minimum (essential). These include the following:

- **AV-1:** Overview and Summary Information.
- **AV-2:** Integrated Dictionary.
- **OV-2:** Operational Node Connectivity Description.
- **OV-3:** Operational Information Exchange Matrix.
- **OV-5:** Operational Activity Model.
- **SV-1:** Systems Interface Description.
- **TV-1:** Technical Standards Profile.

The 26 DoDAF views are designed to document the entire architecture, from requirements to implementation. What is the subset of views from the DoDAF needed to document enterprise architecture? We answer this by building on the work of Sowell [2], Brundage [3], Zachman [4, 5], and Spewak [6]. In a nutshell, the Zachman Framework is an index of architectural information arranged as a five-by-six matrix, documenting a complete architecture. Sowell and Brundage provided a mapping of the DoDAF work products onto this framework. Spewak identified the top two rows of the Zachman Framework as the significant enterprise level information. Leveraging this work collectively, we quickly arrive at

a subset of eight DoDAF views that are required to represent the enterprise architecture. This subset includes the same views as listed in the DoDAF recommended minimal set plus OV-7: Logical Data Model.

Of course, this subset of eight views may be supplemented with additional DoDAF views as required by the specific needs of the enterprise architecture being developed. Still, conspicuously absent are the all-important business, financial, and technical analyses of alternatives – information needed to drive architectural decisions. How do we answer the questions of why one approach or technology was selected over another? Most significantly, how do we illustrate sound business reasons for our decisions? To remedy this, we have enhanced the DoDAF by adding the MV.

The MV draws in part from the Zachman Framework, and also includes the business metrics and investment decision models required to evaluate transition and modernization plans. In short, given the method of enterprise architecture development, in terms of strategies (to-be), baseline (as-is), and transition plan, we need mechanisms and work products to capture the trade-offs, analyses of alternatives, business metrics, financial considerations, and returns on investment to support architectural decision making; we need the MV.

Motivational View

What comprises the MV? Our MV includes the necessary business, financial, and investment models required to evaluate and prioritize the transition alternatives and modernization plans, thus providing a solid business foundation and rationale of why changes need to be made. These models address the issues of metrics, risks, and best value. The work products included are as follows:

- **MV-1:** Business Case.
- **MV-2:** Investment Decision Model.
- **MV-3:** Risk Analysis Model.
- **MV-4:** Best-Value Low-Risk Model.
- **MV-5:** Balanced Scorecard Model.

MV-1: Business Case

The Business Case addresses the rationale for investing the time and resources into making the necessary changes to transform the current as-is to the targeted to-be enterprise architecture. The Business Case starts with the strategies, goals, and objectives regarding how the improvements fit into the enterprise, and why they are significant. The business case also defines the all-important financial rationale measured in dollars and captured as the return on investment (ROI) and break-even time (BET).

The Business Case evaluates a variety of criteria to arrive at a business decision to make the investment required in order to perform the work necessary to gain the projected benefits. Some of the criteria used in crafting a Business Case include business strategic plan, competitive analysis, customer analysis, risk assessment, and financial factors. A sound business plan will illustrate how a reasonable one-time capital expense can achieve a significant recurring cost savings over an acceptable time period. This is the ROI and BET and is typically the driving, if not the only, reason for making changes to the existing architecture. The Business Case is a text document that includes appropriate graphics and financial spreadsheets as needed.

MV-2: Investment Decision Model

The Investment Decision Model provides a mechanism to perform an analysis of cost versus benefit to drive the decision-making process. All alternatives are viewed with regard to costs as compared with benefits. Typical considerations for cost include business process impacts, development method, integration issues, and education needs. Similarly, benefits may include increased capabilities, reduced costs, and productivity improvements.

The decision is driven by the comparison of cost to implement against the business impacts. The positive impacts realized by the business may be in the form of cost avoidance, greater market share, and/or lower risk. Based on the analysis of the supporting data, a model is generated. This model may be viewed as a four-quadrant graph (see Figure 1) labeled as follows: 1-Low Cost/High Benefits, 2-High Cost/High Benefits, 3-High Cost/Low Benefits, and 4-Low Cost/Low Benefits.

MV-3: Risk Analysis Model

The Risk Analysis Model provides a vehicle to identify and analyze risk. Risk is viewed with regard to the probability of occurrence and impact on occurrence. This facilitates a basic three-by-three matrix to evaluate risk (see Figure 2). The table is color-coded and ranges from red to yellow to green. Red indicates high probability of occurrence and high impact on occurrence, and green indicates low probability of occurrence and low impact on occurrence.

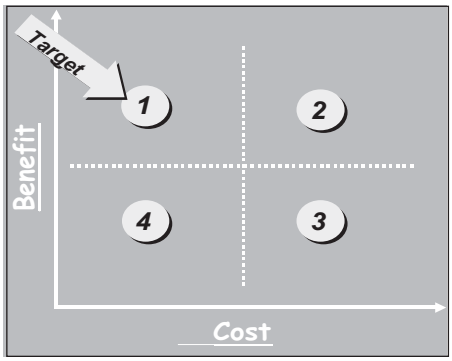


Figure 1: *MV-2: Investment Decision Model*

This is a basic view of the risk table. It may be enhanced to use percentages and/or additional categories as needed. In addition to the table, a risk-mitigation plan is typically generated for each of the identified risks, or a subset (i.e., only high) describing the contingencies when the risk occurs.

MV-4: Best-Value Low-Risk Model

This model provides the next step in selecting the best alternatives by taking a second look at the Investment Decision Model, now comparing the best-value candidates (lowest cost/highest benefits) on the basis of risk. Risk considerations include technology maturity, numbers of interfaces, mission criticality, business process maturity, change management, and training issues. This model is typically represented as a four-quadrant graph labeled as follows: 1-Low Risk/High Value, 2-Low Risk/Low Value, 3-High Risk/High Value, and 4-High Risk/Low Value (see Figure 3 on the next page).

MV-5: Balanced Scorecard Model

The Balanced Scorecard (BSC) [7] is used to provide a common standard model to manage the business and the enterprise architecture, as shown in Figure 4 (see next page). The strength of the BSC is its coupling of leading (operational) and lagging (financial) indicators as well as the alignment of various enterprise capabilities. The BSC integrates various aspects of the enterprise using a set of key performance indicators (KPI).

Figure 2: *MV-3: Risk Analysis Model*

		Probability		
Impacted		Low	Medium	High
	Low	Green	Green	Yellow
	Medium	Green	Yellow	Red
	High	Yellow	Red	Red

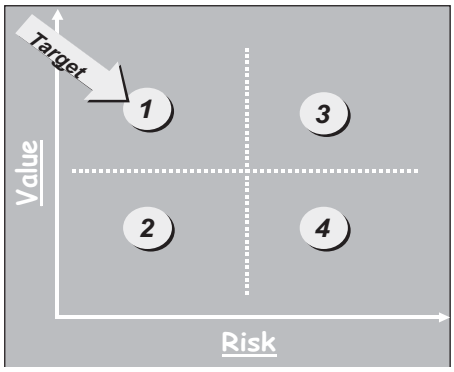


Figure 3: *MV-4: Best-Value Low-Risk Model*

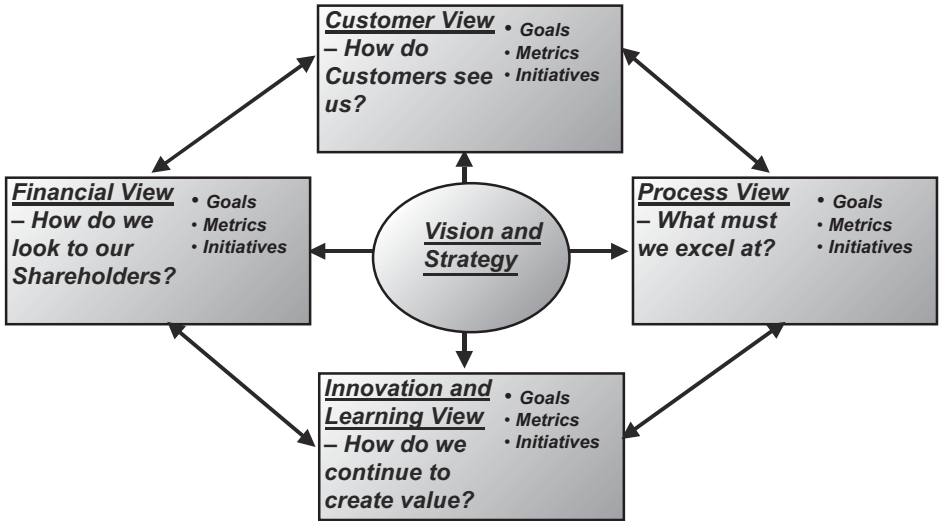
The selection of an appropriate set of KPI is critical. The KPI should map to key business metrics, i.e., the specific measures that drive the business. Examples of these types of metrics for their industries include dollars per flying hour for the airline industry, time to process a claim for the insurance industry, and errors per source line of code for the software industry. It is critical to identify and document the appropriate metrics to use for each aspect of the BSC. These metrics become the basis for declaring success and confirming improvements such as BET, ROI, and other aspects of the business case.

The BSC relates and aligns the enterprise vision and strategy into four views: Customer View, Process View, Innovation and Learning View, and Financial View. By doing this, the model helps facilitate translating the vision and strategy into action. The four views define, describe, and capture the goals, key performance indicators, and initiatives for each of the specific area views thus providing the control and alignment needed to manage the enterprise and supporting architecture.

Summary

The DoDAF views, although adequate for describing enterprise architecture, lack the business perspective needed to develop a sound, transitional plan from as-is to to-be as required in today's architectural projects. The missing business perspective is captured via the addition of the Motivational Views, including the Business Case, Investment Decision Model, Risk Analysis Model, Best-Value Low-Risk Model and Balanced Scorecard Model. These views facilitate the business rationale and trade-offs required to develop a valid and achievable transition plan to transform the enterprise from its current as-is state to the future to-be state. The Motivational View complements the existing DoDAF views, providing a complete holistic view of enterprise architecture.◆

Figure 4: *MV-5: Balanced Scorecard Model*



References

1. Department of Defense Architecture Framework Working Group. "DoD Architecture Framework Ver. 1.0." Washington, D.C.: Department of Defense, Oct. 2001 <http://aitc.aitcnet.org/dodfw>.
2. Sowell, P. Kathie. "The C4ISR Architecture Framework: History, Status, and Plans for Evolution." McLean, Va.: The MITRE Corporation, 1999 <www.mitre.org/work/tech_papers/tech_papers_00/sowell_evolution/sowell_evolution.pdf>.
3. Brundage, George. "Federal Enterprise Architecture Framework Presentation." Washington, D.C.: Department of the Treasury, July 2001 <www.gsa.gov/Portal/gsa/ep/home.do?tabId =0>.
4. Zachman, John A. "A Framework for Information Systems Architecture." IBM Systems Journal 26.3 (1987) <www.research.ibm.com/journal/sj/382/zachman.pdf>.
5. The Zachman Institute for Framework Advancement <www.zifa.com>.
6. Spewak, Steve H. with Steven C. Hill. Enterprise Architecture Planning: Developing a Blueprint for Data, Applications, and Technology. New York: John Wiley & Sons, Sept. 1993.
7. Kaplan, Robert S., and David P. Norton. Translating Strategy into Action: The Balanced Scorecard. Cambridge: Harvard Business School Press, Sept. 1996.

Additional Reading

1. Deming, W. E. Out of the Crisis. Cambridge, MA: Massachusetts Institute of Technology Press, Aug. 2000.
2. Hammer, Michael, and James Champy. Reengineering the Corporation. New York: Harper Collins Publishers, Inc., 1993.
3. Womack, James P., and Daniel T. Jones. Lean Thinking. New York: Simon & Schuster, 1996.

About the Author

D.B. Robi is a Lockheed Martin Qualified Architecture and Certified Lean Six Sigma Black Belt. His career, spanning 19 years, has afforded him opportunities to work in both the commercial and the Department of Defense/federal sectors. Robi has performed as the lead architect on several projects as well as severing as the business architect on efforts in business process modernization, reengineering, and optimization. He is currently the technical lead for Lockheed Martin's Enterprise Architecture Center of Excellence where he leads the development and enhancement of Lockheed Martin's internally developed ARQuest™ Blueprint Process, an approach to developing enterprise architectures. The motivational views described in this article have been incorporated and implemented in ARQuest.

Lockheed Martin Integrated Systems and Solutions (IS&S)
1801 RTE 17C
Owego, NY 13827-3998
Phone: (607) 751-7781
E-mail: dennis.robi@lmco.com

Software Process Improvement – A Good Idea for Other People

I heard a good joke the other day. The story goes that while in a Third World country, a guy grabbed a taxi to go to the airport. At the first red light, the taxi driver just whizzed through without even slowing down. When the passenger complained, the driver said, "Oh, my brother runs red lights all the time. He never has a problem!"

At a stop sign, the driver again whizzed through, never even bothering to look at the cross traffic. When the passenger complained again, the driver replied, "Oh, my brother never bothers to stop at stop signs. He never has a problem!"

Finally, the taxi approached a green light at an intersection. The driver slowed, them came to a full stop, and looked both ways. When the passenger asked why the driver was stopping now, the driver replied, "Well, you never know when my brother is coming! That makes it *our* problem."

The CROSSTALK theme this month is acquisition and supporting articles discuss software process improvement. Now, I'm the first to tell you that I don't really need a process, because I don't have a problem. After all, I'm Dave Cook! You don't know how great I am? Just ask me – or better yet, check out some examples of my work! Best coding you'll ever see. Other programmers blush in shame when they see the craftsmanship of my code. Design? Shoot, I can literally see the interfaces in my head. I'll make the code work, and get it to interface with your substandard code without any problems.

Requirements? Well, you just tell me what you want and I'll write it. You want to change the requirements? Just tell me. I'll fix it. I'm smart enough to understand the rules, and when necessary, to break them. If I violate the configuration management process occasionally, it's because it's in the best interest of the project.

Now, am I really that good? Well, those folks who have had the privilege of working with me will tell you I'm not[1]. And indeed, while I know that I am a good software engineer, I'm almost definitely not as good as I think I am. But the key is, I do think that I am!

Everybody is in favor of process improvement because other people need process discipline. I don't need it,

not me. I'm good. It's the others.

Not that I'm a prima donna – I just think that I'm a darn good developer. All developers think that about themselves. In fact, not a single developer gets up in the morning, looks at himself or herself in the mirror and says, "You know, I'm really not that good." Far from it. If asked, each developer would honestly think they are above average. Many think they are far above average.

Process improvement is something you do to protect yourself from others. If you don't require anybody to manage requirements, record and coordinate designs, etc., then nobody will.

How do you incorporate good software discipline and software processes? Well, I can sure give you some very good hints:

1. Start off with a meeting. Make sure it's two or three hours long. Grab the most boring, monotonous speaker you can find. Spend two or three hours droning on and on about how great the new process is – giving examples that have little or no relevance to what your particular developers do.
2. Tell your developers that they have to follow the process – or else! Give them no way to tailor the process to fit their needs. Don't dream of motivating them; just tell them what to do.
3. Implement the new process immediately throughout the company. For best results, require the use of a new software tool that no one really understands yet. Make sure that no one has more than minimal training on the tool, and that the experts on it are not on-site. In fact, the only available help should be restricted to a poorly designed Web page.
4. As mentioned earlier, make sure that if problems occur, change the work to make it fit the process. After all, the 20-plus years of experience your folks have are no match for a process or tool designed last month by somebody who once took a Java class in college. Don't consider that perhaps common sense is more important than blindly following the process.

Of course these are good hints. Not for you – for me! I'm a consultant, and I can use the work.

See you at the Systems and Software Technology Conference 2004.

— David A. Cook, Ph.D.
Senior Research Scientist
AEgis Technologies Group
dcook@aegistg.com

Can You BACKTALK?

Here is your chance to make your point, even if it is a bit tongue-in-cheek, without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CROSSTALK, we also accept articles for the BACKTALK column. BACKTALK articles should provide a concise, clever, humorous, and insightful article on the software engineering profession or industry or a portion of it. Your BACKTALK article should be entertaining and clever or original in concept, design, or delivery. The length should not exceed 750 words.

For a complete author's packet detailing how to submit your BACKTALK article, visit our Web site at <www.stsc.hill.af.mil>.

1. In fact, the associate publisher of CROSSTALK will probably go out of her way to tell you I'm not. Make sure you go to the Systems and Software Technology Conference 2004 and ask her!

The Door to the CMMI is Waiting for You ...

Do You Have the Right Keys?

If you want your organization to use common, integrated, and improved processes for both Systems and Software, we can help. The Software Technology Support Center will show your organization how to implement the process improvement methodology of the Capability Maturity Model® Integration℠ (CMMI®), which addresses productivity, performance, costs, and stakeholder satisfaction. Make sure you have the right keys. Call us.

® Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Software Technology Support Center
MASE • 6022 Fir Avenue • Building 1238 • Hill AFB, UT 84056 5820
801 775 5555 • DSN 775 5555 • FAX 801 777 8069 • www.stsc.hill.af.mil

Published by the Software Technology Support Center (STSC)

CrossTalk / MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737