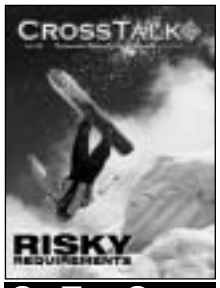
CROSSTALK

April 2002 The Journal of Defense Software Engineering Vol. 15 No. 4

RISKY REQUIREMENTS

Risky Requirements



ON THE COVER
Cover design by Kent Bingham.

Software Engineering Technology

Open Forum

Departments

CROSSTALK

SPONSOR	Lt. Col. Glenn A. Palmer
PUBLISHER	Tracy Stauder
ASSOCIATE PUBLISHER	Elizabeth Starrett
MANAGING EDITOR	Pamela Bowers
ASSOCIATE EDITOR	Julie B. Jenkins
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
CRSIP ONLINE	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 30.

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

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.

Gold Medal Requirements

At the time we prepared this month's issue, the world was coming together for the 2002 Winter Olympics in Salt Lake City. In case you did not know, the CrossTalk staff is located just 30 miles north of Salt Lake City at Hill Air Force Base. For me, it has been very exciting to be in the midst of all of the Olympic activity. Most of all, I'm happy to see the world come together in peace for the games.

During the past few years, Salt Lake City and its surrounding communities have been transformed with improved highways, sports venues, and new businesses in an effort to support the games. As this month's issue highlights requirements and how to avoid the risks associated with specifying poor or invalid requirements, I could not help but think of all of the software requirements associated with putting on the 2002 Winter Olympics.

I recently attended a conference in which an employee of a local telecommunications firm spoke. He identified a myriad of information technologies (IT) employed around each of the Olympic Venues. From computer terminal rooms and networks for media support, to security support, to athlete housing support, to event-ticketing support, and to the events themselves, the software needs associated with hosting the games is absolutely phenomenal. Specifying good and valid requirements to meet the Olympic IT needs could not have been a more important step in making this world event a possibility as well as a huge success.

The defense software world has its own set of challenges when it comes to gathering, specifying, and validating weapon system requirements. Two of the Air Force's Software Technology Support Center's consultants, Theron Leishman and Dr. David Cook, lead off this month's issue with sound advice in R*equirements Risks Can Drown Software Projects*. In this article, you will find helpful tips on how to recognize requirement risks and how to mitigate these risks to keep software projects from going "overboard and drowning" throughout their life cycle.

Next, Dr. Ralph Young discusses *Recommended Requirements Gathering Practices* and explains how requirements elicitation techniques such as interviews, workshops, and storyboards can ensure effective requirements definition and communication among all stakeholders. And in *Reducing Risks Through Proper Specification of Software Requirements*, Al Florence critiques eleven examples of requirement specifications and suggests an improved and less risky way to state these example requirements.

Lastly, be sure to check out our Software Engineering Technology and Open Forum sections this month for more insight into the many challenges and potential risks that defense software projects face today.

I hope that you find this month's issue helpful with techniques and reminders on how to keep from specifying risky requirements. Take the time to make your software project requirements as good as you can, just as if you were training and competing for a gold medal at the Olympics. May we all be winners in the defense software community.

Tracy L. Stauder

Tracy L. Stauder
Publisher

Requirements Risks Can Drown Software Projects

Theron R. Leishman and Dr. David A. Cook
Software Technology Support Center

Software requirements management is often viewed as a stand-alone task in terms of life-cycle activities. Of course, some of the major risks to project completion are incomplete, inaccurate, or vague requirements. In this article we will present and discuss several requirements risks that may have major impacts on the success of software projects. We will then consider strategies to help mitigate the impact of these requirements risks.

A few years ago the movie "Overboard" was released. This is a movie about a rich woman (JoAnna) who was accustomed to having everything her own way. The movie begins with JoAnna hiring an uncouth carpenter (Dean) to remodel the closet of her luxurious yacht. Following several unpleasant encounters between the two during the remodeling project, a major confrontation occurs as the carpenter has completed work, and the yacht is preparing to leave port.

While the carpenter is demonstrating the features of his work, the rich, arrogant, JoAnna asks what the closet is made of. In response, Dean indicates that the closet is made of oak. His response pushes JoAnna over the edge. She says that she wanted the closet to be made of cedar. The carpenter responds that if she wanted the closet to be made out of cedar, she should have asked for cedar. He tells her that he would be glad to make the closet out of cedar, but that his estimate would be more than double because he would have to re-do the whole project. To which she responded, "the whole civilized world knows that closets are made of cedar!" She further indicates that she is not going to pay for "his" mistake! The confrontation escalates to the point that she pushes Dean overboard.

This humorous example demonstrates how easily requirements can be confused between the various stakeholders of any venture. Confusion, misunderstanding, and frustration relative to requirements are major risks to the success of any project.

At the 5th Annual Joint Aerospace Weapons Systems Support, Sensors, and Simulation Symposium in 1999, the results of a study of 1995 Department of Defense (DoD) software spending were presented. A summary of that study is shown in Figure 1. As indicated, of $35.7 billion spent by the DoD for software, only 2 percent of the software was able to be used as delivered. The vast majority, 75 percent, of the software was either never used or was cancelled prior to delivery. The remaining 23 percent of the software was used following modification [1].

A similar study conducted by the Standish Group on non-DoD software projects in 1994 produced very similar results. In over 8,000 projects conducted by 350 companies, only 16 percent of the projects were considered successful.

> *"In over 8,000 projects conducted by 350 companies, only 16 percent of the projects were considered successful ... delivered on time and within budget."*

Success in this study was considered software delivered on time and within budget [2].

More recently, an analysis of the data gathered by the Software Engineering Institute (SEI) on 451 Capability Maturity Model® (CMM®) Level 1 CMM-Based Assessments for Internal Process Improvement conducted from 1997 through August 2001 indicates that requirements continue to be a problem. Of the assessments conducted, approximately 95 percent included an assessment of the Requirements Management Key Process Area (KPA). Of these, only 33 percent fully satisfied the Requirements Management KPA [3].

So what does this data mean? Are we as an industry wasting away billions of dollars due to incompetence? What is the reason for this dismal representation of our capabilities? Further research by the Standish Group indicates the following major reasons for the high failure rate in software development:

- Poor requirements.
- Lack of understanding that cost and schedule are engineering parameters based on requirements.
- Lack of understanding and following a process and a life cycle.

Are we all like Dean, the carpenter in "Overboard"? The above studies indicate that the way we define, analyze, and manage requirements is imposing serious risk to the success of our software projects.

Requirements Risks

But We Gave You Exactly What You Asked For

Have you ever been disappointed when you received exactly what you asked for? The story is told of an executive who when asked about his satisfaction with a new software application indicated that he hated it. When asked why, he responded, "They gave me exactly what I asked for."

The requirements definition phase of a software project is never the self-contained function implied by many software development life-cycle models. The requirements gathering phase is rather an iterative process. It is not enough to obtain the stakeholder's requirements once and assume that they are correct. By so doing, the risk of giving the stakeholder what they ask for, rather than what they really need, is increased.

I Know That I Think I Know What Your Requirements Are

This requirement risk can be exemplified by the co-author's (Leishman's) first software project. Being new to the job, he was determined to demonstrate his abilities. Following a short meeting with the project stakeholders, Leishman disappeared into cubie-land to work his magic. A few weeks later, he emerged from his cubbyhole and

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

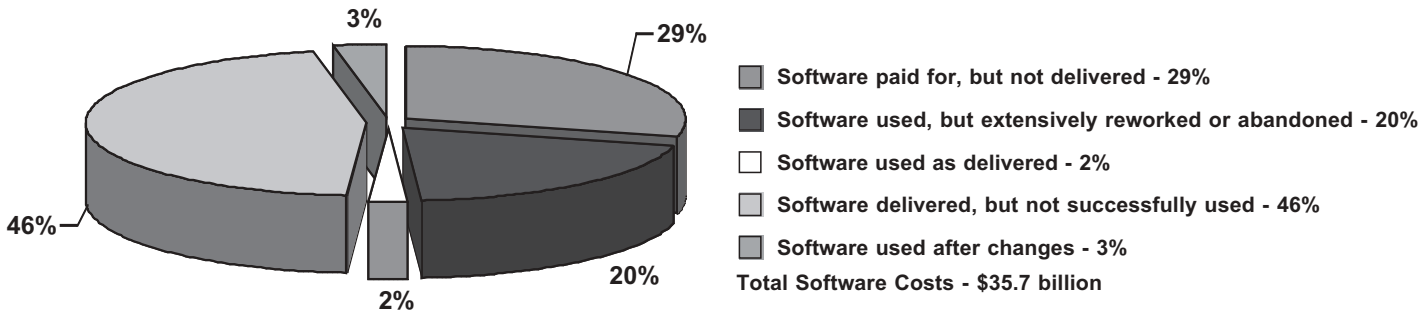


Figure 1: *Findings of a 1995 Department of Defense Software Study*

proudly presented the product of his efforts to the stakeholders. To his dismay, the system did not do what the stakeholders required it to do. They were pleased that their application was delivered on time, but very upset that the application did not do what they required.

This risk is a very common occurrence. It is characterized by not involving project stakeholders throughout the development effort. Typically this requirements risk will not be identified until stakeholder testing or implementation. By not taking necessary steps to assure that we understand the requirements, we are inviting project rework that will result in schedule delays and cost overruns.

Overboard Assumptions

This is the requirements risk we identified in the movie example in the introduction of this article. There are things in each of our frames of reference that appear to be no-brainers! In the movie, JoAnna indicated that, "The whole civilized world knows that closets are made of cedar." Why then was there a question of the customer's requirements? Because the frame of reference of the rich and famous was totally different than that of the country hick carpenter.

The risk of assuming that developers and customers have the same thoughts about system requirements is like assuming that we all agree on political or religious issues. As in the case of the carpenter, these issues are often not identified until the customer first sees the application. At this stage of development, there will almost certainly be negative impacts on both project cost and schedule.

The Expectations Cloud

A few years ago, a woman responded to an advertisement to have the carpets in her house cleaned. The advertisement offered three rooms of carpet cleaned for $69.95. Seeing this as a good deal, she contacted the company and indicated that she would like the three-room special. The representative came to her house and cleaned the carpet in the three rooms identified. Upon completion, the technician presented a bill for $249. Needless to say, the woman was not a happy camper! When questioned about the bill, the technician indicated that the three-room special applied to rooms with dimensions of 10 feet by 10 feet. The three rooms cleaned were each considerably larger than this.

So, who was at fault in this carpet-cleaning situation? Was it the woman's responsibility to assure that there was agreement to the expectations prior to the work being completed? Or was it the responsibility of the service provider to assure that he met the woman's expectations? Regardless of where the responsibility for clarification belonged, the woman was a dissatisfied customer!

There are numerous software project examples of stakeholder expectations not being resolved prior to the project beginning. Like the carpet example, if the customers' expectations are not met, future services will not be requested.

The Never-Ending Requirements Story

The world we live in is rapidly changing. The business, economic, political, and military environments are all changing and fluctuating daily based on world conditions. Technology is likewise changing at a very rapid rate. In this environment, it is unrealistic to assume that software requirements are not going to change during the development process.

The risk of requirements changing is that the changes will spiral out of control and prevent an application from ever being completed. The never-ending requirement story is also one of the never-completing software project. An additional risk of constant change is the lack of common requirements understanding by the various stakeholders during the various iterations of requirements. The result is confusion, chaos, misunderstanding, and software either never being completed, or if completed never being used.

I Don't Know What I Want, but I'll Know It When I See It

There are also those occasions when stakeholders are not interested in clearly defining their requirements. These stakeholders assume they have the luxury of having developers play the development game over and over until they get something they like.

The risk of allowing this attitude to exist in the stakeholder environment is that valuable development resources are wasted playing a guessing game. Every iteration expended and left floundering in the dark to arrive at a solution that meets the stakeholders' unknown requirements increases development costs and extends the project schedule.

A further risk with this attitude is that the software project is not perceived to belong to the stakeholders. They remain removed from the project, do not accept ownership, and thus increase the risk of project failure.

Rapid Development Requirements Risks

In an attempt to increase the speed of software development, various rapid development approaches have been devised. The use of Rapid Application Development, Unified Modeling Language Use Cases, and Extreme Programming are examples of new approaches that have been taken to speed software development.

When conducted properly, each of these approaches includes a process of requirements analysis. Often these new approaches have sought newer, better ways to document, verify, and track stakeholder requirements. The risk associated with these approaches is the same old temptation to cut corners when conducting requirements analysis that we often find in more traditional approaches. Often, developers who are not properly trained in these approaches assume that "rapid" means incomplete or haphazard. Some do not recognize the need for adequate requirements analysis and will elimi-

nate or minimize the requirements analysis process. This is a tremendous risk to project success.

Failure to Recognize that Faulty Requirements Represent a Risk

According to the SEI Software Capability Maturity Model® (SW-CMM®), "Requirements management involves establishing and maintaining an agreement with the customer on the requirements for the software project. The agreement forms the basis for estimating, planning, performing, and tracking the software project's activities throughout the software life cycle [4]." Years of experience have revealed that errors occurring in the requirements stage of the development process turn out to be the most difficult and costly to fix.

The Software Technology Support Center has, upon the request of various program managers, conducted Independent Expert Program Reviews. These reviews are conducted to assist program managers in determining the wellness of their programs and evaluating areas of concern to the program. A common finding in many of these reviews is that requirements elicitation, analysis, and management is being conducted in an inefficient manner. In several of these cases, requirements inadequacies have proven to be a major contributor to the program being behind schedule and over budget.

It is essential to understand that requirements, or more appropriately the lack of adequate requirements, can be a significant risk to the success of any project. The soundness of the organization's requirements management process should be taken into consideration when evaluating the risk that requirements may have on the success of your project.

Can't You Read My Mind?

Failure to document requirements in a format that promotes clear, complete, and comprehensive understanding is a serious risk to project success. In his article, "When Telepathy Won't Do: Requirements Engineering Key Practices [5]," Karl Wiegers indicates that the most essential and yet often neglected practice is to write down, or document, the requirements. He goes on to indicate that requirements should be documented in some acceptable, structured format as they are gathered and analyzed.

After all, is not the purpose of the requirements process to communicate a shared understanding of what the system requires among the project stakeholders? The need to communicate requirements is directly connected with the need to document requirements. If requirements are not documented in some manner, it is impossible for multiple individuals to come to a common understanding and agreement of the requirements.

Strategies to Mitigate Requirements Risks

Develop and Follow Sound Processes and Procedures

A software process can be defined as a set of activities, methods, practices, and transformations that people employ to develop and maintain software and the associated products [4]. The problem comes when the process that I follow and the process others in the organization follow is not the same. The result is a lack of consistency in the way software is developed and maintained. This lack of consistency leads to confusion, misunderstandings, development delays, and cost overruns.

"The need for user involvement is imperative. By requiring user involvement in the requirements process, the risk to the project of inadequate requirements is greatly reduced."

An important step to mitigating requirements risks is for the organization to develop and strictly follow sound processes and procedures relative to requirements engineering. These processes and procedures should include direction to developers in the following areas:

- Requirements elicitation.
- Requirements analysis.
- Documentation of requirements.
- Requirements verification, review, and approval.
- Configuration control of requirements.
- Requirements traceability.

The organization should also ensure that roles and responsibilities relative to these processes and procedures are clearly defined. In one aerospace software development organization we know, the application development manager went to great lengths to impress upon the software project leaders that the ultimate success of each development project rested upon their heads. The responsibility for resolving assumptions, ambiguities, and clarifying stakeholder expectations rests upon the developers.

Incorporate Requirements into All Software Life Cycles

By now, the need and value of having organizationally accepted software life-cycle models and methods should be well established. From our research and knowledge of various life-cycle models, they all include a requirements analysis, requirements management, or stakeholder requirements phase.

By assuring that the life cycle(s) approved for usage within the organization require proper levels of requirements administration, the risk associated with projects relative to requirements will be reduced.

Provide Training to Those Responsible for Requirements Management

Requirements elicitation, analysis, documentation, verification, and maintenance are not simple tasks. The ability to facilitate the elicitation of requirements and follow the process through to completion requires knowledge and skill. Those within the organization who are responsible for assuring that requirements are managed, must receive the training and mentoring necessary to provide them with the ability to fulfill this responsibility.

Require User Involvement

In their CHAOS Report of 1995, the Standish Group indicated that information technology (IT) projects fail because they lack user involvement, follow incomplete requirements and specifications, and experience confusion caused by changing requirements and specifications. The lack of adequate user involvement is a virtual guarantee of project failure.

In the article, "13 Common Objections Against User Requirements Analysis, and Why You Should Not Believe Them [6]," the author looks at 13 excuses used for not involving users in the development of Web-based projects. The conclusion drawn is that user involvement is imperative to the success of IT projects even in the rapidly changing web environment.

Users are the primary stakeholders in

most software projects. To assume that the primary stakeholder can be eliminated and have the project succeed is approaching lunacy. Yet as indicated in the study conducted by the Standish Group, lack of user involvement in requirements analysis and verification is the root cause of many development project problems.

The need for user involvement is imperative. By requiring user involvement in the requirements process, the risk to the project of inadequate requirements is greatly reduced. This mitigation strategy combined with having and following sound processes and procedures, which are also supported by defined software life cycles, will greatly reduce requirements risks.

Always Document Requirements

The need to reach a common understanding of requirements among key project stakeholders is vital to the success of a software project. To reach common understanding it is essential that requirements be documented. Any text on software requirements will indicate that good requirements' characteristics include the following:

- Correct.
- Complete.
- Consistent.
- Unambiguous.
- Verifiable.
- Understandable.
- Traceable.
- Modifiable.

Historically, requirements have been documented in a System Requirements Specification (SRS) or other similar document. In recent years requirements documentation has taken on various forms. Today there are three basic forms of documentation used for requirements: text based, box based, and graphics based.

Text-based documentation relies on formally defined language to describe system requirements. This approach has been criticized as being old fashioned, slow, and subject to various interpretations based on the background of the various stakeholders.

Box-based documentation uses geometric symbols to represent various aspects of the system requirements. This approach is designed with the software engineer in mind and is generally more comfortable for the software engineer to follow and understand. This approach is traditionally more difficult for end users to understand because of the learning curve associated with this type of documentation.

Graphic-based documentation is the more recent of the three documentation forms. It was developed to support object-oriented development and design techniques. This method uses geographic symbols to represent the actual objects within a system. As with the box-based methods, graphic-based documentation is tailored more toward the developer than to the end user's understanding.

We recommend that some combination approach be adopted. We have experience using a combination of graphics and text. This approach helps the users insert themselves into the process or requirement being documented. It also is easily understood by the developers and serves as a useful tool to both elicit and validate requirements.

Various programming methodologies such as Extreme Programming and the Unified Systems Development Process are typically considered design and coding aids. These approaches recommend proper requirements gathering, analysis, and validation be conducted during software development and maintenance. These methodologies document requirements using models, diagrams, and text in an environment of heavy stakeholder involvement. These requirements can and should be maintained during various iterations of development and their traceability maintained through the entire development process.

Validate Software Requirements

Requirements validation is an essential step to ensure that requirements are properly understood and documented. This mitigation strategy goes hand in hand with the need for dedicated stakeholder involvement. As requirements are documented, the stakeholders should validate them. Often the act of requirements validation will uncover requirements issues that can be discovered in no other way.

Hold Formal Requirements Reviews

It has consistently been proven that it costs more to correct requirements errors that are discovered later in the life cycle. For example, a requirement error caught during design might cost four times more to correct than if the error had been discovered and corrected during the requirements phase itself [7]. Studies indicate that document reviews greatly reduce the errors in critical documents.

Errors, inconsistencies, ambiguities, and confusion can be greatly reduced by holding formal reviews of software requirements documents. Teams performing requirements management activities should be trained in sound review methods. Formal reviews will greatly improve the quality of software requirements documents.

Strictly Manage Requirements Changes

Strategies exist to help manage requirements "creep." Such strategies consist of good configuration management, formal reviews of change requests, and a formal change control process. Requirements creep should be around 1 percent per month. In fact, creep of more than 2 percent a month is probably a sure sign of a project that will never reach completion. Without sound strategies for managing change, a project will fail.

Even with sound strategies, stakeholders must be aware of the high cost and high risk of change. For the good of the project, some changes are simply too expensive or too difficult. Such changes must be postponed until after a version of the product is successfully delivered.

Summary

Requirements management is critical to the success of any software development or acquisition project. It requires the ability to deal with stakeholders of various backgrounds with various goals, interests, and objectives. By their very nature, there are risks associated with the elicitation, analysis, and validation of requirements. If not given proper attention, these requirements' risks can push software projects overboard and result in software projects drowning!

By recognizing the potential impact of these requirements' risks, steps can be taken to turn these risks into strengths. Instead of requirements being the source of problems, a disciplined software requirements management process can help to assure the success of your software projects.◆

References

1. Jarzombek, Stanley J. "The 5th Annual Joint Aerospace Weapons Systems Support, Sensors, and Simulation Symposium (JAWS S3)." Proceedings, 1999.
2. The Standish Group International, Inc. The CHAOS Report, 1994.
3. Software Engineering Institute. "Process Maturity Profile of the Software Community." Mid-year Update, Aug. 2001.
4. Software Engineering Institute. The Capability Maturity Model Guidelines for Improving the Software Process. Boston: Addison-Wesley, 1994.

COMING EVENTS

April 9-10
Southeastern Software Engineering Conference
Huntsville, AL
www.ndia-tvc.org/SESEC2002/

April 29-May 2
Software Technology Conference 2002 "Forging the Future of Defense Through Technology"
Salt Lake City, UT
www.stc-online.org

May 13-17
Software Testing Analysis and Review (STAREAST 2002)
Orlando, FL
www.sqe.com/stareast

June 3-6
Combat Identification Systems Conference
Colorado Springs, CO
www.usasymposium.com

June 3-7
Software Quality Engineering Test Week
Chicago, IL
www.sqe.com

June 17-19
11th Annual Executive Forum on Modeling and Simulation
Norfolk, VA
www.dmso.mil/index.php?page=27

July 22-25
Joint Advanced Weapons Systems Sensors, Simulation, and Support Symposium (JAWS S3)
Colorado Springs, CO
www.jawswg.hill.af.mil

5. Wiegers, Karl E. "When Telepathy Won't Do: Requirements Engineering Key Practices." Cutter IT Journal May 2000.
6. D'Hertefelt, Sim. "13 Common Objections Against User Requirements Analysis, and Why You Should Not Believe Them." Interaction Architect.com 9 June 2000.
7. Boehm, Barry, and Wilfred J. Hansen. "The Spiral Model as a Tool for Evolutionary Acquisition." CROSSTALK May 2001.

Additional Reading

1. Dorfman, Merlin. "Requirements Engineering." SEI Interactive Mar. 1999.
2. Kar, Pradip, and Michelle Bailey. "Characteristics of Good Requirements." INCOSE Symposium, 1996.
3. Thomas, Bill. "Meeting the Challenges of Requirements Engineering." SEI Interactive Mar. 1999.
4. VanBuren, Jim, and Dr. David A. Cook. "Experiences in the Adoption of Requirements Engineering Technologies." CROSSTALK Dec. 1998.
5. Wiegers, Karl E. Software Requirements. Microsoft Press, 1999.
6. Wiegers, Karl E. "Karl Wiegers Describes 10 Requirements Traps to Avoid." Software Testing and Quality Engineering Jan./Feb. 2000.

Did this article pique your interest?

You can hear more from these authors at the Fourteenth Annual Software Technology Conference Apr. 29-May 2, 2002, in Salt Lake City, UT. They will be presenting in Track 7 on Wednesday, May 1, at 10:00 a.m.

About the Authors

Theron R. Leishman is a consultant currently on contract with the Software Technology Support Center at Hill Air Force Base, Utah. Leishman has 18 years experience in various aspects of software development. He has successfully managed software projects and performed consulting services for the Department of Defense, aerospace, manufacturing, health care, higher education, and other industries. This experience has provided a strong background in systems analysis, design, development, project management, and software process improvement. Leishman has a master's in business administration from the University of Phoenix. He is a Level II Certified International Configuration Manager (CICM) by the International Society of Configuration Management (ISCM), and is employed by TRW.

Software Technology Support Center
7278 4th Street
Bldg. 100 G19
Hill AFB, UT 84056
Phone: (801) 775-5738
Fax: (801) 777-8069
E-mail: theron.leishman@hill.af.mil

David A. Cook, Ph.D., is the principal engineering consultant, Shim Enterprises, Inc. He is currently assigned as a software-engineering consultant to the Software Technology Support Center at Hill Air Force Base, Utah. Dr. Cook has more than 27 years of experience in software development and software management. He was formerly an associate professor of computer science at the U. S. Air Force Academy (where he was also the department research director) and also the deputy department head of the Software Professional Development Program at the Air Force Institute of Technology. Dr. Cook has published numerous articles on software process improvement, software engineering, object-oriented software development, programming languages, and requirements engineering. He has a doctorate degree in computer science from Texas A&M University, and he is an authorized Personal Software Process instructor.

Software Technology Support Center
7278 4th Street
Bldg. 100
Hill AFB, UT 84056
Phone: (801) 775-3055
Fax: (801) 777-8069
E-mail: david.cook@hill.af.mil

Recommended Requirements Gathering Practices

Dr. Ralph R. Young
Northrop Grumman Information Technology

This article provides suggested conditions for performing requirements gathering and recommended requirements gathering practices. The author has conducted an extensive review of industry literature and combined this with the practical experiences of a set of requirements analysts who have supported dozens of projects. The sidebar on page 10 summarizes a set of recommended requirements gathering practices. Involving customers and users throughout the development effort results in a better understanding of the real needs. Requirements activities should be performed throughout the development effort, not just at the beginning of a project.

A "requirement" is a necessary attribute in a system, a statement that identifies a capability, characteristic, or quality factor of a system in order for it to have value and utility to a user [1]. According to Steve McConnell in *Software Project Survival Guide*, "The most difficult part of requirements gathering is not documenting what the users 'want'; it is the effort of helping users figure out what they 'need' that can be successfully provided within the cost and schedule parameters available to the development team."[1]

Each requirement should be necessary, verifiable, attainable, unambiguous, complete, consistent, traceable, concise, implementation-free, and have a unique identifier [1]. All of these characteristics of a good requirement should be self-evident, with the possible exception of "implementation-free." The reason that a requirement should be implementation-free is that requirements specify "what" shall be provided and not "how" – the how is a design aspect rather than a requirement. Documenting the rationale for each requirement (why it is required) is a good technique to reduce the number of requirements. Taking this one step, according to industry requirements consultant Ivy Hooks' experience, can eliminate "up to half" of the stated requirements [2].

Begin by understanding the organization's "business requirements." This leads to a "vision and scope" document that describes the background leading to the decision to develop a new or modified system or capability and describes the system to be developed. An agreed upon understanding of the capability is critical to a successful project. Consider having iterative scoping meetings with customers and users. The process of requirements elicitation itself generates more detailed and creative thinking about the problem that in turn can affect the scope. As the possibilities for a solution emerge, there are numerous decision points concerning what should and should not be included within the scope of the system.

The next step is to gather the stated requirements of the customers and users of the new capability. An effective requirements practice distinguishes "stated" requirements from "real" requirements [1]. Industry experience has shown that customers and system developers should jointly evaluate stated requirements to ensure that each is a verified need.

Part of the requirements process is to prioritize requirements.[2] This is important, because rarely is there enough time and money to provide everything that is wanted. It is also beneficial to focus on product benefits, not features [3]. Benefits refer to the necessary requirements. Adding unnecessary features adds design constraints and increases costs.

"Using peer reviews, scenarios, and walk-throughs to validate and verify requirements results in a more accurate ... specification and higher customer satisfaction."

It is estimated that 85 percent of the defects in developed software originate in the requirements [1]. Once defects are embedded in the requirements, they tend to resist removal. They are especially difficult to find via testing. Therefore it is crucial that training be required for requirements analysts and engineers that explains how to reduce the common types of requirements errors, including incorrect assumptions (49 percent), omitted requirements (29 percent), inconsistent requirements (13 percent), and ambiguities (5 percent) [2].

Use peer reviews and inspections to reduce defects in all your requirements representations. Peer reviews and inspections are a best practice way of eliminating defects. I recommend a peer review of all work products. The extent of the review should be based on the criticality of the work product. Peer reviews are a very effective method for reducing the costs of a project because they identify defects earlier. Rework is estimated at 45 percent of project costs, industry-wide [1]. Using peer reviews, scenarios, and walk-throughs to validate and verify requirements results in a more accurate requirements specification and higher customer satisfaction.

Inspections are a very rigorous form of peer reviews and should be considered for requirements representations. Gilb and Graham provide an excellent guide for inspections of any type of document [4]. According to Gilb, the capability to perform Gilb inspections requires five days of formal training and a lot of rigor.[3] One advantage of the Gilb approach is that he advocates "sampling" of work products rather than review of the entire product – the idea is that by identifying defects in the first few pages, the author can utilize this feedback to address similar problems throughout the document or work product.

Some believe that all requirements should be listed in a requirements document such as a Software Requirements Specification. Experience has shown that it is helpful to think of "several" artifacts comprising your requirements specification: the database in your automated requirements tool, the vision and scope statement for the project, the requirements document, and other requirements lists or descriptions provided by customers and users. These can include lists of requirements met by related legacy (historical) systems and the list of system-level (real) requirements evolved by the requirements manager/requirements engineer. This enables us to have a more comprehensive understanding of the real requirements that

Recommended Requirements Gathering Practices

The following is a list of recommended requirements gathering practices. They are based on the author's extensive review of industry literature combined with the practical experiences of requirements analysts who have supported dozens of projects.

1. Write and iterate a project vision and scope document.
2. Initiate a project glossary that provides definitions of words that are acceptable to and used by customers/users and the developers, and a list of acronyms to facilitate effective communication.
3. Evolve the real requirements via a "joint" customer/user and developer effort. Focus on product benefits (necessary requirements), not features. Address the minimum and highest priority requirements needed to meet real customer and user needs.
4. Document the rationale for each requirement (why it is needed).
5. Provide training for requirements analysts and selected customer/user representatives that explains the following:
   - The role of the requirements analyst, e.g., to evolve real requirements working with customers and users, not to invent requirements independently or to "gold plate."
   - How to write good requirements.
   - The types of requirements errors and how these can be reduced.
   - The value of investing more in the requirements process.
   - The project and/or organization's "requirements process."
   - Overview of the methods and techniques that will be used.
   - How to use the project's automated requirements tool.
   - The role of validation and verification during requirements definition.
6. Establish a mechanism to control changes to requirements and new requirements.
7. Prioritize the real requirements to determine those that should be met in the first release or product and those that can be addressed subsequently.
8. When the requirements are volatile (and perhaps even when they are not), consider an incremental development approach. This acknowledges that some of the requirements are "unknowable" until customers and users start using the system.
9. Use peer reviews and inspections of all requirements work products.
10. Use an industry-strength automated requirements tool.
    - Assign attributes to each requirement.
    - Provide traceability.
    - Maintain the history of each requirement.
11. Use requirements gathering techniques that are known, familiar, and proven in the organization such as requirements workshops, prototyping, and storyboards.
12. Provide members of the project team (including requirements analysts) who are domain/subject matter experts.
13. Evolve a project and organizational approach based on successful use of policy, process, methods, techniques, and tools. Provide a mechanism such as working groups to share information and "best practices" among projects.
14. Establish a continuous improvement ethic, teamwork approach, and a quality culture.
15. Involve customers and users throughout the development effort.
16. Perform requirements validation and verification activities in the requirements gathering process to ensure that each requirement is testable.

is communicated effectively to all stakeholders.

One of our most common problems is taking on too much work – attempting to exceed requirements rather than addressing the minimum requirements to meet real needs. Thus, meeting minimum requirements is in the customers' best interests. It helps avoid the problems of late deliveries, budget overruns, low morale, and poor quality [5].

Preferred Requirements Gathering Techniques

Following are a set of recommended requirements elicitation techniques. Among almost 40 such techniques available [1], only a few have proven most effective. These techniques can be used in combination. Their advantages are that they are effective in emerging the real requirements for planned development efforts. Kotonya and Sommerville [6] provide a good discussion of the context for requirements elicitation and analysis. More detailed discussions of these techniques are provided in Leffingwell and Widrig [7] and in Sommerville and Sawyer [8].

Interviews. Interviews are used to gather information. However, the predisposition, experience, understanding, and bias of the person being interviewed influence the information obtained. The use of context-free questions by the interviewer helps avoid prejudicing the response [9]. A context-free question is a question that does not suggest a particular response. For example, who is the client for this system? What is the real reason for wanting to solve this problem? What environment is this product likely to encounter? What kind of product precision is required?

Document Analysis. All effective requirements elicitation involves some level of document analysis such as business plans, market studies, contracts, requests for proposals, statements of work, existing guidelines, analyses of existing systems, and procedures. Improved requirements coverage results from identifying and consulting all likely sources of requirements [10].

Brainstorming. Brainstorming involves both idea generation and idea reduction. The goal of the former is to identify as many ideas as possible, while the latter ranks the ideas into those considered most useful by the group. Brainstorming is a powerful technique because the most creative or effective ideas often result from combining seemingly unrelated ideas. Also, this technique encourages original thinking and unusual ideas.

Requirements Workshops. Requirements workshops are a powerful technique for eliciting requirements because they can be designed to encourage consensus concerning the requirements of a particular capability. They are best facilitated by an outside expert and are typically short (one or a few days). Other advantages are often achieved – participant commitment to the work products and project success, teamwork, resolution of political issues, and reaching consensus on a host of topics. Benefits of requirements workshops include the following:

- Workshop costs are often lower than are those for multiple interviews.
- They help to give structure to the

requirements capture and analysis process.
- They are dynamic, interactive, and cooperative.
- They involve users and cut across organizational boundaries.
- They help to identify and prioritize needs and resolve contentious issues.
- When properly run, they help to manage user's expectations and attitude toward change [11].

A special category of requirements workshop is a Joint Application Development (JAD) workshop. JAD is a method for developing requirements through which customers, user representatives, and developers work together with a facilitator to produce a requirements specification that both sides support. This is an effective way to define user needs early. Wood and Silver in *Joint Application Development* [12] assert that quality systems can be built in 40 percent less time utilizing JAD. They explain how to perform JAD and provide diagrams, forms, and a sample JAD design document.

Prototyping. Prototyping is a technique for building a quick and rough version of a desired system or parts of that system. The prototype illustrates the capabilities of the system to users and designers. It serves as a communications mechanism to allow reviewers to understand interactions with the system. See Sommerville's *Software Engineering* [13] for a good discussion of prototypes and how they can be used. Prototyping sometimes gives an impression that developers are further along than is actually the case, giving users an overly optimistic impression of completion possibilities. Prototypes can be combined effectively with other approaches such as JAD and models.

Use Cases. A use case is a picture of actions a system performs, depicting the actors [14]. It should be accompanied by a textual description and not be used in isolation of other requirements gathering techniques. Use cases should always be supplemented with quality attributes and other information such as interface characteristics. Many developers believe that use cases and scenarios (descriptions of sequences of events) facilitate team communication. They provide a context for the requirements by expressing sequences of events and a common language for end users and the technical team.

Be cautioned that use cases alone do not provide enough information to enable development activities. Other requirements elicitation techniques should also be used in conjunction with use cases. Requirements consultant Ivy Hooks recommends using operational concepts as a simple, cost-effective way to build a consensus among stakeholders and to address two large classes of requirements errors: omitted requirements and conflicting requirements [2]. Operational concepts identify user interface issues early, provide opportunities for early validation, and form a foundation for testing scenarios in product verification.

Storyboards. A storyboard is a set of drawings depicting a set of user activities that occur in an existing or envisioned system or capability. Storyboards are a kind of paper prototyping. Customers, users, or developers start by drawing pictures of the screens, dialogs, toolbars, and other elements they believe the software should provide. The group continues to evolve these until real requirements and details are worked out and agreed upon. Storyboards are inexpensive and eliminate risks and higher costs of prototyping. Another related technique is storytelling: the writing of vignettes to envision new products and services based on perceived user needs and the possibilities offered by emerging technologies.

Interfaces Analysis. Missing or incorrect interfaces are often a major cause of cost overruns and product failures. Identifying external interfaces early clarifies product scope, aids risk assessment, reduces product development costs, and improves customer satisfaction. The steps of identifying, simplifying, controlling, documenting, communicating, and monitoring interfaces help to reduce the risk of problems related to interfaces. Hooks and Farry provide a thorough discussion and recommendations [2].

Modeling. A model is a representation of reality that is intended to facilitate understanding. The CORE requirements tool has behavioral modeling capabilities. Behavior is allocated to physical components of the planned system. See Vitech's Web page at <www.vtcorp.com> for information concerning this tool and a trial version that can be downloaded. Uses of the tool for modeling and example problems are described in Buede [15]. In a recent study of 15 requirements engineering teams supporting relatively small projects (average of 10 person-years of effort with project duration of 16.5 months), use of prototypes and models helped eliminate ambiguities and inconsistencies and correlated with the most successful projects [10].

Performance and Capacity Analysis. Hofmann and Lehner [10] provide an insight based on their study of 15 requirements engineering efforts: Stakeholders emphasized that concentrating on system functions and data resulted in a lack of attention to the total system requirements and in incomplete performance, capacity, and external interface requirements. Thus, it is vital to ensure that the requirements gathering process provides for all requirements (requirements coverage).

An Innovative Concept

For an innovative approach to gathering requirements, see "A Quick, Accurate Way to Determine Customer Needs [16]." The authors of this article believe customers tend to say one thing during requirements elicitation and then do something entirely different. They feel that this problem is largely due to reliance on traditional requirements gathering approaches such as focus groups, surveys, and interviews that do not deal effectively with contradictions in peoples' responses.

The authors advocate "a new technology" called imprint analysis. Imprint refers to the collection of associations and emotions unconsciously linked with a word, concept, or experience. They believe this method produces findings that remain consistent over time because it takes human emotions into account. Emotion is the trigger to action; emotions in the present dictate peoples' emerging needs. The authors believe that imprint analysis can actually forecast customer behavior.

A Cautionary Note

Everyone involved in a particular project should use a common set of methods and techniques. To that end, it is advisable to have project discussions and training sessions to evolve the desired "project approach." Projects should use methods and techniques that have been used successfully on previous projects in that organization. If there is no local precedent, hire staff from outside the organization who have previous successful experience. And above all, I strongly recommend that the project involve people who have previously successfully used all methods and techniques that are to be employed. Providing formal training for developers who are expected to use new methods and tools is a valuable investment.

Automated Requirements Tools

I recommend the use of an automated requirements tool to support a development effort of any size. Tiny projects might get away with using Microsoft Word or Microsoft Excel; however, most proj-

ects require an industry-strength requirements tool such as DOORS, Requisite Pro, or Caliber RM with capabilities that extend beyond "requirements management."

Using a requirements tool facilitates requirements elicitation because it enables better understanding of the requirements by both the customer and the developer. Also, an effective requirements tool helps prioritize requirements, provides requirements traceability throughout the development effort, allows assignment of multiple attributes (characteristics of requirements) to all requirements, and facilitates managing requirements changes [1].

Conclusions and Recommendations

There is a wealth of information and guidance available in back issues of CROSSTALK, books, articles, and industry conference publications, and also from the "lessons learned" on projects in our own organizations. Much has been written, but perhaps too little has been conscientiously applied on actual projects. Do not try to do everything at once. Rather, encourage the project team to select and commit to a few improved practices that make sense in your environment.

Establish a few useful metrics that enable evaluation of implementation effectiveness and institutionalization of selected practices. Remember, the things that are measured and tracked are the ones that improve. Make a concerted effort to improve project communications as well as teamwork. A committed, highly motivated team can accomplish most anything.◆

References

1. Young, Ralph R. Effective Requirements Practices. Boston: Addison-Wesley, 2001. See also <ralph young. net>, a Web site devoted to requirements-related topics.
2. Hooks, Ivy F., and Kristin A. Farry. Customer-Centered Products: Creating Successful Products Through Smart Requirements Management. New York: AMACOM (publishing arm of The American Management Association), 2001.
3. Smith, Preston G., and Donald G. Reinertsen. Developing Products in Half the Time. 2nd ed. New York: John Wiley & Sons, Inc., 1998.
4. Gilb, Tom, and Dorothy Graham. Software Inspection. Reading, Mass.: Addison-Wesley, 1993. See also <www.result-planning.com>.
5. Whitten, Neal. "Meet Minimum Requirements: Anything More Is Too Much." PM Network Sept. 1998.
6. Kotonya, Gerald, and Ian Sommerville. Requirements Engineering: Processes and Techniques. Chichester, England: John Wiley & Sons, 1998.
7. Leffingwell, Dean, Don Widrig, and Edward Yourdon. Managing Software Requirements. Boston: Addison-Wesley, 2000.
8. Sommerville, Ian, and Pete Sawyer. Requirements Engineering: A Good Practice Guide. New York: John Wiley & Sons, 1997.
9. Gause, Donald C., and Gerald M. Weinberg. Exploring Requirements: Quality Before Design. New York: Dorset House Publishing, 1989.
10. Hofmann, Hubert F., and Franz Lehner. "Requirements Engineering as a Success Factor in Software Projects." IEEE Software July/Aug. 2001: 58-66.
11. Graham, Ian. Requirements Engineering and Rapid Development: An Object-Oriented Approach. Reading, MA: Addison-Wesley, 1998.
12. Wood, Jane, and Denise Silver. Joint Application Development. New York: John Wiley & Sons, 1995.
13. Sommerville, Ian. Software Engineering. 6th ed. Harlow, England: Addison-Wesley, 2001.
14. Schneider, Geri, Jason P. Winters, and Ivar Jacobson. Applying Use Cases: A Practical Guide. Reading, Mass.: Addison-Wesley, 1998.
15. Buede, Dennis M. The Engineering Design of Systems: Models and Methods. New York: John Wiley & Sons, 2000.
16. Afors, Cristina, and Marilyn Zuckerman Michaels. "A Quick, Accurate Way to Determine Customer Needs." Quality Progress, July 2001, 82-87.
17. McConnell, Steve. Software Project Survival Guide. Redmond, Wash.: Microsoft Press, 1998.
18. Wiegers, Karl E. "First Things First: Prioritizing Requirements." Software Development Magazine Sept. 1999: 24-30.

Notes

1. Adapted from Steve McConnell, *Software Project Survival Guide* [17]. See Chapter 8 for valuable suggestions concerning requirements development.
2. Visit Karl Wiegers' Web site <process impact.com/goodies.shtml> to download a Microsoft Excel spreadsheet useful for prioritizing requirements. See also Wiegers' article, *First Things First: Prioritizing Requirements* [18].
3. Industry consultant Robert Sabourin trains and facilitates Gilb inspections. He advises that the basic training can be accomplished in four hours, including one example inspection. A mentor or champion is required to train moderators, scribes, and process administrators. Sabourin's experience is that Gilb inspections provide good value, for example, to inspect requirements against sources and to inspect all downstream work from requirements. Performing inspections can foster communication and gain buy-in. Inspections can be used to test artifacts that otherwise would be nearly impossible to test objectively. Inspections can be implemented with minimal impact on the normal workflow. See <www. amibug.com>.
4. See Chapter 3 of Graham's *Requirements Engineering and Rapid Development* for a detailed discussion of organizing and running workshops.

About the Author

Ralph R. Young, DBA, is the director of Software Engineering, Systems and Process Engineering, Defense Enterprise Solutions at Northrop Grumman Information Technology, a leading provider of information technology and systems-based solutions. Dr. Young leads a requirements working group of requirements engineers. He teaches a 10-hour Requirements Course for Practitioners and consults frequently concerning both requirements engineering and process improvement. Dr. Young has received awards for teamwork, leadership, continuous improvement, and publishing, and is often recognized for his contributions in process management and improvement. He is the author of *Effective Requirements Practices.*

Northrop Grumman
Information Technology
Mail Stop 5S3
1500 PRC Drive
McLean, VA 22102
Phone: (703) 556-1030
E-mail: young_ralph@prc.com

Reducing Risks Through Proper Specification of Software Requirements

Al Florence
MITRE Corp.[1]

Requirement definition, specification, analysis, and validation and verification are some of the biggest challenges faced by software engineers. In many software requirements documentation, the requirements are ambiguous and inconsistent. They may not be uniquely identified, making them untraceable and difficult to test. In many cases, they are specified at a level too high or too low at the system or at the design level, not at the software requirements level. If these challenges are addressed, the risk of developing systems that do not satisfy requirements will be mitigated.

This article presents several examples that address the challenges faced by individuals specifying software requirements. For instance, while redeveloping legacy systems, a government agency reverse engineered the existing software requirements. With knowledge of the application domain, several teams reverse engineered and defined the requirements. They represented the user, the contractors, and the acquisition organization. This author was assigned as a consultant to guide the teams in the proper specification of requirements. The requirements were analyzed and validated against the following critical attributes:

- Complete: Requirements should be as complete as possible. They should reflect system objectives and specify the relationship between the software and the rest of the subsystems.
- Traceable: Each requirement must be traceable to some underlying source such as a system-level requirement. Each requirement should have a unique identifier allowing the software design, code, and test procedures to be precisely traced back to the requirement.
- Testable: All requirements must be testable to demonstrate that the software end product satisfies its requirements. To be testable, requirements must be specific, unambiguous, and quantitative whenever possible. Vague, general statements must be avoided.
- Consistent: Requirements must be consistent with each other; no requirement should conflict with any other requirement. Check requirements by examining all requirements in relation to each other for consistency and compatibility.
- Feasible: It must be feasible to develop software that will fulfill each software requirement. Requirements that have questionable feasibility should be analyzed during requirements analysis to prove their feasibility. If they cannot be implemented they should be eliminated.
- Uniquely Identified: Uniquely identifying each requirement is essential if requirements are to be traceable and are able to be tested. Uniqueness also helps in stating requirements in a clear and consistent fashion.
- Design Free: Software requirements should be specified at the requirements level and not at the design level. Describe the software requirement functionally from a requirement point of view, not from a software-design point of view, i.e., describe the system functions that the software must satisfy. A requirement reflects "what" the software shall accomplish while the design reflects "how" the requirement is implemented.
- Using "Shall" and Related Words: In specifications, using the word "shall" indicates a binding provision, i.e., one that must be implemented by the specification users. To state non-binding provisions, use "should" or "may." Use "will" to express a declaration of purpose (e.g., "The government will furnish ...") or to express future tense [1].

> *"Each requirement should have a unique identifier allowing the software design, code, and test procedures to be precisely traced back to the requirement."*

If projects allocate sufficient time and effort to validate requirements against these critical attributes during their definition and specification, projects will mitigate the risks associated with inadequate requirements.

Requirement Effort Examples

The following examples represent several legacy systems that were in the process of redevelopment in a modernization effort. They depict the requirements effort only and do not reflect any other life-cycle activities: design, implementation, test, or operation. These examples show some of the requirements as initially specified by the teams, followed by this author's critique of the requirements against the critical attributes, and finally the resulting re-specification.

Example 1

Initial specification: Software will not be loaded from unknown sources onto the system without first having the software tested and approved.

Critique: If the software is tested and approved, can it be loaded from unknown sources? If the source is known, can it be loaded if it has not been tested and approved? This requirement is ambiguous, which makes it difficult to implement and test. It is stated as a negative requirement making it difficult to implement. A unique identifier is not provided, which makes it difficult to trace. The word "shall" is missing.

Re-specification: 3.2.5.2 Software shall be loaded onto the operational system only after it has been tested and approved.

Example 2

Initial specification: 3.4.6.3 The system shall prevent the processing of duplicate electronic files by checking a new SDATE record. An e-mail message shall be sent.

Critique: There are two "shalls" under one requirement number. This is a vague requirement. What is the e-mail message? The requirement has design implications [SDATE record]. A requirement should specify what the data in the record are and not the name of the record. The name of

the record should appear in the design and code not in the requirement. As specified it cannot be implemented or tested.

Re-specification: 3.4.6.3 The system shall:

a. Prevent processing of duplicate electronic files by checking the date and time of the submission.
b. Send the following e-mail message:
   1. Request updated submission of date and time, if necessary, or
   2. That the processing was successful, when successful.

Example 3

Initial specification: 3.2.5.7 The system shall process two new fields (provides production count balancing info to the states) at the end of state record.

Critique: This requirement cannot be implemented or tested. It is incomplete. What are the two new fields? "Info" should be spelled out.

Re-specification: 3.2.5.7 The system shall provide the following data items (provides production count balancing information to the states) at the end of state record:

a. SDATE record.
b. YR-TO-DATE-COUNT.

Re-Critique: This rewrite has design implications [SDATE record and YR-TO-DATE-COUNT]. A requirement should specify what the data in the record are and not the name of the record.

Re-specification: 3.2.5.7 The system shall provide the following data items (provides production count balancing information to the states) at the end of state record:

a. Submission date and time.
b. Year to date totals.

Example 4

Initial specification: 3.2.5.9 All computer-resident information that is sensitive shall have system access controls to ensure that it is not improperly disclosed, modified, deleted, or rendered unavailable. Access controls shall be consistent with the information being protected and the computer system hosting the data.

Critique: Two "shalls" under one identifier thus two requirements. The requirement is vague and incomplete. What does "consistent" mean? The requirement needs to identify the sensitive information. As specified, it cannot be implemented or tested.

Re-specification: 3.2.5.9 All sensitive computer-resident information shall have system access controls consistent with the level of protection required to ensure that the information is not improperly disclosed, modified, deleted, or rendered unavailable. (Reference Sensitive Information Table 5.4.1 and Levels of Protection for Sensitive Information Table 5.4.2.)

Example 5

Initial specification: 3.3.2.1 The system shall have no single point failures.

Critique: This is an ambiguous requirement. It needs definition and/or identification of what components and/or functions the "no single point failures" applies. As specified it cannot be implemented or tested.

Re-specification: 3.3.2.1 The following system components shall have no single point failure:

a. Host servers.
b. Networks.
c. Network routers.
d. Access servers.
e. Hubs.
f. Switches.
g. Firewalls.
h. Storage devices.

Example 6

Initial specification: 3.2.7.1 The system shall purge state control records and files that are older than the operator or technical user-specified retention period.

Critique: This requirement cannot be implemented or tested as stated. It is vague without specifying the retention period or providing a reference as to where the information can be obtained.

Re-specification: 3.2.7.1 The system shall purge state control records and files that are older than the retention period that is input into the system by either:

a. The operator.
b. The technical user.

Example 7

Initial specification: 3.2.6.3 The system shall receive and process state data from the State Processing Subsystem. The system shall provide maintenance of the state data files and generate various reports.

Critique: There are two "shalls" under one requirement number and multiple requirements in the specification. The word "process" in the first "shall" is vague. The requirement needs to define the processing required. The second "shall" does not provide for valid requirements; they cannot be implemented or tested as stated. The requirement needs identification of type/amount of maintenance required. The term "various reports" is ambiguous.

Re-specification: 3.2.6.3 The system shall receive:

a. Production data that contain data from multiple states.
b. Financial state data for one or more states, extracted by the State Processing Subsystem.

3.2.6.4 The system shall parse multi-state data to respective state files.

3.2.6.5 The system shall display a summary screen reporting the results of processing for each state containing:

a. State totals.
b. State generic totals, and
c. State unformatted totals

Example 8

Initial specification: 3.2.7.1 The system shall not prevent individuals from entering the year for which they intend the payment, but shall provide a checkpoint for them to ensure that they are not making a mistake in entering the correct year.

Critique: This is a negative requirement; negative requirements should not be specified. They cannot be implemented. A requirement should have all conditions that are required. If conditions are not required they will not be implemented. There are two "shalls" under one requirement number. I suggest that this requirement be structured in a positive fashion.

Re-specification: 3.2.7.1 The system shall:

a. Allow individuals to enter the payment year.
b. Provide a checkpoint to ensure that individuals enter the correct payment year.

Example 9

Initial specification: 3.2.7.3 After the system receives the validation file, the system shall:

- Notify the individual about acceptance or rejection.
- The acceptance file must contain the name control and ZIP code of the approved individual.
- Rejected validation request must include the Reason Code.

Critique: The second and third bullets do not make sense, try to read them without the first bullet:

- The system shall the acceptance file must...
- The system shall rejected validation…

The requirement uses a "shall" and a "must." Unique identifiers are not provided. The requirement uses bullets, which should not be used in specifying requirements. Bullets cannot be traced. This requirement is ambiguous and cannot be implemented or tested.

Re-specification: 3.2.7.3 When the system receives a validation file the system shall:

a. Reject the file if it does not contain the approved individual's:

1. Name.
2. ZIP code.

b. Notify the individual about acceptance or rejection with a reason code. (Reference Reason Codes Table 5.4.8.)

Example 10

Initial specification: 3.2.8.2 The enrollment process shall take from one (1) to ten (10) calendar days to complete for all enrollment types.

3.2.8.3 The enrollment process shall take no more than three (3) days to complete for:

a. Credit enrollment.
b. Note enrollment.

Critique: These two requirements are inconsistent and in conflict with each other.

Re-specification: 3.2.8.2 The enrollment process shall take:

a. One (1) to three (3) calendar days to complete for:
   1. Credit enrollment.
   2. Note enrollment.
b. One (1) to ten (10) calendar days to complete for all other enrollment types.

Example 11

Initial specification: 3.2.8.6 When doing calculations the software shall produce correct results.

Critique: Really? This is not a requirement. This type of requirements should not be specified. It should be deleted.

Re-specification: Requirement deleted.

Conclusion

The teams identified more than 1,000 requirements. The issues with their initial specification represented the entire spectrum of the critical attributes: complete, traceable, testable, consistent, feasible, uniquely identified, and design free. The teams were receptive to the critiques, resolved issues, and implemented the recommendations willingly. The requirements resulting from this effort were reviewed with senior management, accepted as specified, baselined, and allocated to development teams for implementation.

If sufficient time and proper effort is taken to validate requirements against critical attributes during their definition and specification, software projects will mitigate the risks associated with requirements and will considerably improve their probability of success. It is a well-known fact that if this is not done, projects pay the consequences during implementation and integration and test, not to mention during operation.◆

Reference

1. Military Standard Specification Practices. MIL-STD-490A. U.S. Department of Defense, 4 June 1985.

Note

1. The views expressed are those of the author and do not reflect the official policy or position of the MITRE Corporation.

Suggested Readings

1. IEEE Std. 830-1998. IEEE Recommended Practices for Software Requirements Specifications. IEEE Computer Society. 20 Oct. 1998.
2. Cook, David A., and Les Dupaix. "The Requirements for Good Requirements." Software Technology Conference Proceedings. Mar. 2001.

About the Author

Al Florence has been employed at major technology firms and is currently at the MITRE Corporation. He has been involved in all phases of the life cycle as a manager and a developer, from concept to retirement in different engineering disciplines, including systems, software, test, configuration management, quality assurance, and process improvement as a developer and as a manager. His work has involved many diversified projects: spacecraft, aircraft, missiles, weapon systems, particle accelerators, simulation, and information systems. He has been involved with the definition, specification, and validation of requirements on many of these projects. Florence has a bachelor's of science degree in mathematics and physics from the University of New Mexico, and he did graduate work in computer science at the University of California in Los Angeles and at the University of Southern California.

The MITRE Corporation
7515 Colshire Drive
McLean,VA 22102-3481
Phone: (703) 883-7476
Fax: (703) 883-1339
E-mail: florence@mitre.org

Seven Characteristics of Dysfunctional Software Projects

Michael W. Evans, Alex M. Abela, and Thomas Beltz
Integrated Computer Engineering , Inc.

Taking advantage of its many years of experience in identifying and evaluating project risks in large-scale software systems acquisition and development programs, Integrated Computer Engineering has developed a risk database. Their analysis of this risk database has identified seven predominant characteristics that provide insight into the causes of dysfunctional software projects. This article identifies these characteristics and the typical real-world risks that accompany each.

Integrated Computer Engineering (ICE), Inc., a subsidiary of American Systems Corporation, is experienced in identifying and evaluating project risk in large-scale systems acquisition and development programs. During the last 12 years, ICE assessed more than 280 federal, state, Department of Defense, and commercial software-intensive programs and projects. These projects were responsible for the acquisition or development of leading-edge weapons, communications, financial, logistics, and public service automated data processing systems.

The project risks that were collected during ICE's 12 years of project assessments began to form a substantial database of useful information. Also added to this risk database are project risks gathered from risk studies conducted by the Institute for Defense Analysis [1], risks identified by Capers Jones [2] and Tom DeMarco [3], and from risks identified during risk management services ICE performed in support of the Software Program Managers Network [4]. To date, the ICE risk database has grown to more than 800 primary and secondary project risk indicators.

What emerged from the ICE risk database were seven predominant characteristics relating directly or indirectly to common failures observed among those system acquisition and development projects that had the greatest difficulty delivering a quality product on time and on budget. The seven common characteristics are listed below:

1. Failure to Apply Essential Project Management Practices. Many troubled projects fail to apply proven project management disciplines like cost estimation, project scheduling, resource planning, configuration management, and proactive risk management, then wonder why their project is in constant turmoil.
2. Unwarranted Optimism and Unrealistic Management Expectations. Some managers recognize the potential for negative impact on their project from potential problem areas; however, they choose to see things through rose-colored glasses, assuming that problems will work themselves out even when all available evidence raises the red flag.
3. Failure to Implement Effective Software Processes. Many projects fail to implement effective software processes because their approach to process application is not balanced. Some apply minimal process and rely on staff expertise, while others insist on rigorous global process conformance.

> *"Managing a software project requires 'courageous' and often clairvoyant individuals ... willing to confront today's challenges to avoid tomorrow's catastrophes."*

4. Premature Victory Declarations. Pressures to deliver timely products often result in premature declarations of completion by managers. Success cannot be declared until products have been completed with the built-in contracted quality and reliability.
5. Lack of Program Management Leadership. Managing a software project requires "courageous" and often clairvoyant individuals who are willing to confront today's challenges to avoid tomorrow's catastrophes. We have observed two types of problem managers: those with software engineering and no management experience, and those with management and no software engineering experience. Both types lack the ideal blend of both technical and managerial know-how.
6. Untimely Decision-Making. Some managers avoid making time-critical decisions until it is too late, even when they are faced with overwhelming warning signs of impending problems.
7. Lack of Proactive Risk Management. Many projects claim to implement risk management but few do so effectively. "What distinguishes the best organizations and best managers is not just how well they do in their successful efforts, but how well they contain their failures [5]."

Now, let's take a closer look at some real-world risks that have been associated with each of the seven characteristics.

Failure to Apply Essential Project Management Practices

Typical risks are as follows (risk designators are listed in Table 1, page 19):

- (A) The process being followed and decisions being made will result in a product that may not satisfy the critical needs of the user and are inconsistent with the severity of the consequences of project failure.
- (I) Project plans do not describe how technology will be used resulting in a need to continuously rework inconsistent products and correct resulting problems.
- (J) Software reliability problems will not be discovered because procedures are not established for the collection and analysis of error data generated during software development.
- (C) Project plans are unrealistic or not implemented and do not result in a predictable development environment.
- (K) Software defects will not be found because the contractor has neither conducted nor planned for software design inspections or walkthroughs.
- (L) Essential system functions do not perform adequately or reliably due to

testing problems or insufficient testing of key software components.

What we repeatedly find through assessments is that while the mainstream software tasks have been reasonably well planned and implemented, certain essential project management practices are not. The practices that are routinely at the bottom of list are: cost estimation, scheduling, resource planning, configuration management, risk management, earned value reporting, performance-based metrics, re-estimation, quality assurance, and rigorous testing.

Some managers perceive these practices as bureaucratic red tape that only gets in the way of real engineering. Also, methods such as risk management, metrics, and re-estimation often provide managers with more reality than they care to know or handle.

Unwarranted Optimism and Unrealistic Executive Management Expectations

Typical risks are as follows:

- (A) The process being followed and decisions being made will result in a product that may not satisfy the critical needs of the user and are inconsistent with the severity of the consequences of project failure.
- (B) The staff is not capable of implementing the product and applying the technologies selected. Excessive turnover may impact project success.
- (C) Project plans are unrealistic or not implemented and do not result in a predictable development environment.

In some projects there is an underlying belief that all will be well. The reality is that planning for the worst and being surprised when it does not occur is a much more effective way to manage a software project. In 1995, only 16 percent of software projects were expected to finish on time and on budget. An estimated 53 percent of projects cost nearly 190 percent of their original estimates [6]. When managing or participating in a system acquisition or development project there is absolutely no rationale for optimism. Historical data do not support an overly confident posture when managing large complex high-tech programs. Why then, is this unfettered optimism so common?

Two principal causes of unwarranted optimism have been observed. The first relates to the second degree of ignorance, or "not knowing what you don't know." Staff members with insufficient experience may have unrealistic optimism about success for the following reasons:

- They are not aware of the magnitude of the tasks or the problems they are attempting to solve.
- They oversimplify what it will take to achieve the required result or product.
- They attempt to implement silver-bullet technology solutions without having thoroughly evaluated their effectiveness or impact on the program.

The second cause of unwarranted optimism stems from unrealistic executive management expectations. "There is a major cultural barrier to accurate estimation [and scheduling] that must be highlighted … If an early estimate [or schedule] predicts higher cost, longer schedules, or lower quality than client or manager expectations, there is a strong tendency to challenge the validity of the estimate. What often occurs in this situation is that the project manager is directed to recast the estimate so that it falls within preset and arbitrary boundary conditions [7]."

This self-imposed cultural barrier that some executive managers place between themselves and their program managers forces those managers to report unrealistic estimates, schedules, and project risks to their customers and to oversight organizations.

"The reality is that planning for the worst and being surprised when it does not occur is a much more effective way to manage a software project."

The cost of runaway or defective systems often gets personalized in the dismissal or demotion of the responsible executive.

> I don't want yes men around me. Tell me what you think even if it costs you your job.
>
> – Louis B. Mayer, legendary head of production at MGM

With the threat of removal hanging over their heads, many managers establish a "can-do-at-all-cost" mentality. Bad news is not tolerated, projections of problems are not acceptable, and anything other than full steam ahead is punished in the severest manner.

Failure to Implement Effective Software Processes

Typical risks are as follows:

- (G) The technical process being used is inconsistent with the project's requirements and the staff's ability to implement it.
- (D) Design and code defects will not be discovered until late in the development – too late to avoid cost, schedule, or quality impacts.
- (H) The design (system or software depending on where the indicator is observed) may not support the application's critical safety or security requirements.
- (F) There is inefficient software development due to failure to allocate requirements early in the design phase.

Many software projects' managers assume that since trained software engineers staff the project, project-specific standards, guidelines, and common tools are unnecessary.

There are two factors at work here that impact the ability of the project to apply common processes to specific projects. The first is project uniqueness. To paraphrase Tim Lister, each project is unique [8]. Each has its own quirky clients, its own unique staff, and its own expectations of success. Could it be that adaptation of process is 90 percent of the problem and the common processes are marginal? Technology and process are not a "cookie-cutter" solution to every development problem; the key to success is adaptation of the technology and process to meet the unique challenges of a specific project or program.

"With the right people you might succeed without process maturity, but … the best process in the world will not make you successful without the right people [9]."

The second factor is project balance. Technology, tools, processes, and people must all be in balance at the project, not the organizational level.

Premature Victory Declarations

Typical risks are as follows:

- (L) Essential system functions do not perform adequately or reliably due to testing problems or insufficient testing of key software components.
- (N) The system may not satisfy the needs or expectations of the user when delivered.

- (O) Early release of unqualified products results in unexpected failures, failures in key user areas, and potentially corrupted data, which destroys confidence in future releases.

We have observed that the pressure to deliver timely product has a tendency to overwhelm the need for quality and consequently results in an early and unwarranted victory declaration by management. "… Some of my staff would tinker on a task forever, all in the name of quality. But, in some cases the market doesn't care that much about quality. Instead, it's screaming for the product to be delivered yesterday and is willing to accept it even in a quick and dirty state. The decision to pressure people into delivering a product that doesn't measure up to their own quality standards is almost always a mistake [9]."

A clear understanding of the customer's quality expectations is an essential prerequisite to client satisfaction. Delivery of a faultless solution that is significantly over-budget and so late in delivery as to be obsolete will fail to satisfy a customer as much as a product delivered on time that does not meet specified requirements or that has poor reliability.

Lack of Program Management Leadership

Typical risks are as follows:

- (C) Project plans are unrealistic or not implemented and do not result in a predictable development environment.
- (E) The planning has not been updated recently and now is out of date with the project environment and does not reflect current agreements or constraints.
- (M) Customer relationships cause an environment that is unstructured and precludes successful implementation of a product within cost and schedule.
- (B) The staff is not capable of implementing the product and applying the technologies selected. Excessive turnover may impact project success.

"Poor project management will defeat good engineering, and is the most frequent cause of project failure [10]." Too many people who have never developed software are making decisions about how software should be developed. Attributes of a good software project manager include a broad range of technical software development experience, the ability to manage people and the dynamics of a team environment, and the willingness to proactively manage project risk and make timely decisions. To paraphrase Tom DeMarco, "Managers … make the craziness go away [11]."

During a recent assessment of a severely troubled software project, the managers did not know the status of the product, the staff was demoralized, and the project was severely over budget and behind schedule. What we discovered will amaze you. Management had set a goal for this site to become Software Engineering Institute, Capability Maturity Model® (CMM®) Level 3 compliant by a certain deadline. Unbelievably, they diverted 40 percent of the experienced engineering staff to work the CMM issue. The manager said, "We thought the rest could take up the slack." This isn't rocket science. If you want to manage a software project you have to hunker down and do it right: no shortcuts, no nonsense, no silver bullets; just a laser beam to the finish line.

Untimely Decision-Making

Typical risks are as follows:

- (D) Design and code defects will not be discovered until late in the development – too late to avoid cost, schedule, or quality impacts.
- (E) The planning has not been updated recently and now is out of date with the project environment and does not reflect current agreements or constraints.
- (F) There is inefficient software development due to failure to allocate requirements early in the design phase.

"Management is the art of planning work so that it can be accomplished within constraints of time, cost, and other resources at a level that will be competitive in the marketplace [12]." Delays caused by slow decision making erode these constraints even further while the project team waits for clear direction or crucial resources. Unless plans remain current, a project can be caught off guard when unexpected problems arise, leaving managers with insufficient controls, discipline, and/or support facilities to make effective and informed decisions.

A second problem results from late decision making. Simply stated, you can fix a bad decision, but no action occurs while projects wait for managers to decide what to do. During many project assessments we conducted, engineers have stated that they knew what actions to take and were ready to proceed, but could not move out until management decided the prudent course.

"Fast decision-makers often make better decisions than slow decision-makers," according to a study by Kathleen Eisenhardt of Stanford University and Jay Bourgeois of Virginia University. Fast decision-makers "set up systems to collect a range of information on their business and markets constantly, and then make decisions using the data available. Slow decision-makers first analyze a problem and sort out the questions that must be answered. Only then, do they go out and look for that information [13]."

Lack of Proactive Risk Management

Typical risks are as follows:

- (P) Miscellaneous risks not being tracked make project success unlikely.
- (Q) Risk management may not prove effective or identify key risks.
- (R) The lack of an effective risk management process results in unplanned problems impacting the project.

"The problem of project management, like that of most management [is] to find an acceptable balance among time, cost, and performance [14]." When a project moves out of balance, a risk results. Often schedule performance becomes the most important issue due to customer pressures, resulting in a loss of focus on cost and product performance and increased project risk. "An effective risk management program is dynamic and ongoing throughout the development process and requires the participation of everyone involved [15]."

During our assessments, a significant amount of time is spent on determining the effectiveness and degree to which a project implements risk management as part of its management structure. In our experience, this assessment area is a bull's-eye indicator of the potential for overall project risk. Projects that fail to do an effective job of managing risk are constantly reacting to problems, while those that manage risk well anticipate rather than react. "Your organization will be much better once it moves away from reacting to change, and toward proactive anticipation and management of change [16]." To maximize potential for success, risk management should play a visible and key role in the process of project management.

"Risk management transcends modern management theory, such as Total Quality Management and Business Process Reengineering, because it is basic to decision making. Risk management is based on theories that provide different strategies for decision making under problematic conditions [17]."

Analysis

Table 1 shows each of the seven risk characteristics and their respective risk designators (an upper case letter over the range A to R). Each risk event in the ICE database was characterized against the risk designators, and the tallied results are shown in the

third column from the left. The risk designator events were accumulated for each risk characteristic (fourth column), and the frequency of occurrence relative to all observed events in the database was calculated (far right column). It should be noted that the percentages in the far right column do not total 100 percent, as the risk designators are not unique to each characteristic.

The ranking of the characteristics is by frequency of characteristic occurrence; therefore, the data show what may be the likely dysfunctional causes, but not their relative impact on projects or programs.

Characteristic	Risk Designator	Number of risk events applicable to specific Risk Designator	Number of risk events for Characteristic	Frequency of occurrence relative to all observed risk events (See Note 1)
1. Failure to Apply Essential Project Management Practices.	A	246	480	57%
	I	6		
	J	36		
	C	66		
	K	10		
	L	116		
2. Unwarranted Optimism and Unrealistic Executive Management Expectations.	A	246	344	41%
	B	32		
	C	66		
3. Failure to Implement Effective Software Processes.	G	162	248	30%
	D	15		
	H	26		
	F	45		
4. Premature Declarations of Victory.	L	116	165	20%
	N	46		
	O	3		
5. Lack of Program Management Leadership.	C	66	106	13%
	E	3		
	M	5		
	B	32		
6. Untimely Decision-making.	D	15	63	8%
	E	3		
	F	45		
7. Lack of Proactive Risk Management.	P	4	24	3%
	Q	9		
	R	11		

Note 1: The total number of risk events categorized (841 events) was used as the baseline population of risk events for frequency of occurrence calculations.

Table 1: *Seven Risk Characteristics*

Conclusion

When reviewing dysfunctional software projects, a reasonable approach would be to consider the risk descriptions for each of the seven characteristics we have identified and determine whether they apply.

Why do projects not address these issues if they are so apparent? The first reason is denial. When you are fighting the day-to-day realities of a software project, it is very easy to assume that the indicators of disaster are probably wrong, and the project will not be impacted the way the other 12 projects were. Denial is the excuse that enables program managers to make dumb decisions.

The second reason is cultural barriers. Coincidentally, all of the seven factors we identified focus on cultural, rather than technical, issues. "Since 1979 we have been contacting whoever is left on the project staff to find out what went wrong. For the overwhelming majority of the bankrupt projects we studied, there was not a single technological issue to explain the failure [18]." Factors such as the seven we addressed here do matter, and they should be considered essential components of any project.◆

References

1. Technical Risk Indicators for Embedded Software Development. Institute for Defense Analysis, Paper P-3027, Oct. 1994.
2. Jones, Capers T. Assessment and Control of Software Risks. New Jersey: Yourdon Press, Feb. 1994.
3. DeMarco, Tom. Why Does Software Cost So Much? New York: Dorset House Publishing, 1995.
4. Software Program Managers Network. 16 Critical Software Practices For Implementing Performance-Based Management. Ver. 3.0, Arlington, Va.: Integrated Computer Engineering, Inc., 2 Aug. 2000.
5. DeMarco, Tom. Why Does Software Cost So Much? New York: Dorset House Publishing, 1995. 62.
6. Standish Group International. "Chaos." Open Computing Copyright, Mar. 1995 SPC.
7. Jones, Capers T. Assessment and Control of Software Risks. New Jersey: Prentice Hall, Feb. 1994. 158.
8. Lister, Tim. "Software Management for Adults." Software Technology Conference, 1996.
9. Davis, Alan. "Software Lemmingineering." IEEE Software Sept. 1993.
10. Humphrey, Watts. "Three Dimensions of Process Improvement: Part I: Process Improvement." CROSSTALK Feb. 1998.
11. DeMarco, Tom. Why Does Software Cost So Much? New York: Dorset House Publishing, 1995. 66.
12. Putnam, Lawrence H., and Ware Meyers. Industrial Strength Software, Effective Management Using Measurement. Los Alamitos, Calif.: IEEE Computer Society Press, 1996. 1.
13. Putnam, Lawrence H., and Ware Meyers. Industrial Strength Software, Effective Management Using Measurement. Los Alamitos, Calif.: IEEE Computer Society Press, 1996. 13.
14. P.V. Norden. Useful Tools For Project Management, Operations Research in Research and Development. Edited by B. V. Dean. New York: John Wiley & Sons, 1963.
15. Molt, George. "Risk Management Fundamentals in Software Development." CROSSTALK Aug. 2000.
16. Boehm, Barry, Raymond Madachy, and Chris Abts. "Future Trends: Implications in Cost Estimation Models." CROSSTALK Apr. 2000.
17. Hall, Elaine. Managing Risk. Reading Mass.: Addison-Wesley 1997. 5.
18. DeMarco, Tom, and Tim Lister. Peopleware, Productive Projects and Teams 2nd ed. New York: Dorset House Publishing, 1999. 4.

Did this article pique your interest?

You can hear more from Michael Evans at the Fourteenth Annual Software Technology Conference Apr. 29-May 2, 2002, in Salt Lake City, UT. He will be presenting in Track 6 on Thursday, May 2, at 1:00 p.m.

About the Authors

Michael W. Evans, former owner and president of Integrated Computer Engineering, (ICE) Inc., is now a senior vice president with American Systems Corporation, which recently acquired ICE as a wholly owned subsidiary. He has led more than 250 program-risk assessments of large federal, Department of Defense (DoD), and commercial software acquisition and development projects and is considered an expert in software testing, quality assurance, and configuration management. He is co-founder of the Software Program Managers Network, the driving force behind the DoD's Software Acquisition Best Practices Initiative. His published books include *Principles of Productive Software Management*, *Productive Test Management*, *Software Quality Assurance and Management*, and *The Software Factory*.

Integrated Computer Engineering, Inc.
142 North Central Avenue
Campbell, CA 95008
E-mail: candca@aol.com

Alex M. Abela has 18 years of professional engineering experience, with more than seven years experience as a senior project leader of major Defense programs. He has worked in Defense Research and Development organizations in Australia, United Kingdom, and the United States. His engineering experience spans the disciplines of information technology, electronics, electro-optics, and telecommunications. His interests in information technology include the provision of technical advice on software systems best practices. Currently he is working for the Australian Department of Defense as senior technical specialist in surveillance and reconnaissance systems.

Australian Department of Defense
Land Engineering Agency
DPM-4-32, Defense Plaza
661 Bourke Street
Melbourne, Victoria
Australia 3000
Phone: +61 3 9622 2945
Fax: +61 3 9622 2782

Thomas Beltz serves as executive assistant at American Systems Corporation. He has more than 10 years of experience in Department of Defense and commercial software acquisition, operational test and evaluation (OT&E), and software best practices implementation. At the U.S. Navy OT & E Force, he authored an OT&E Task Schedule with more than 1,000 program-tracking controls to determine system readiness for formal operational evaluation. He co-developed Integrated Computer Engineering's Software Development Capability Evaluation Tool, providing a quantitative assessment of a developer's capability to build software while meeting program life-cycle requirements, and he has participated in more than 25 independent risk assessments of large-scale software development programs.

ICE Directorate EA
Phone: (757) 463-8483, ext. 21
E-mail: thomas.beltz@iceincusa.com

Add Decision Analysis to Your COTS Selection Process

Barbara Cavanaugh Phillips and Susan M. Polen
Software Productivity Consortium

Processes for evaluating, comparing, and selecting commercial off-the-shelf (COTS) products are often nonexistent, or when performed, are vague, poorly documented, nonrepeatable, and inconsistent. At best, these deficiencies can decrease confidence in the selection decision; at worst, this lack of forethought leads to poor decisions that delay a project's development, increase life-cycle costs, and reduce quality. It is critical to a project's success that the most appropriate COTS product is selected. This article describes an answer to this common problem and a decision-making process specifically for COTS evaluation, and it provides some lessons learned from the application of this process.

How can you select commercial off-the-shelf (COTS) software from a market of more than 50 available options? Which option has the best value for your project's features? Should you believe everything the vendor says? How can you account for that when making decisions? How can you convince your company that you have little confidence in decisions made quickly based on a few data points? When is it valuable to follow a structured process? What process and method should you follow? How can you save money and time in future evaluations?

In response to its members' need for a repeatable and systematic process for evaluating and selecting components and COTS software, the Software Productivity Consortium (the Consortium) developed the Comparative Evaluation Process (CEP). This article describes the Consortium's process so that you may add some ideas to your own process.

It is critical to a project's success that the most appropriate or "right" component be selected. Both the budget and schedule of a project are affected if the right component is not selected at the start. Off-the-shelf component use and integration are often required by clients and are written into contracts. The CEP is an instance of the Decision Analysis and Resolution (DAR) process area of the Capability Maturity Model® Integration℠ (CMMI℠) and a structured decision-making process. The relationship between CMMI's DAR and CEP is discussed at the end of this article. The process provides detailed guidance to alleviate the typical problems that occur during an evaluation. For example, managers often find that evaluations never end and are quite costly to the program. CEP's first activity is to scope the effort and schedule the activities.

℠ CMM Integration and CMMI are service marks of Carnegie Mellon University.

Several features separate CEP from similar processes. One feature is a suggested set of criteria that expands the understanding and evaluation of characteristics beyond commonly evaluated characteristics such as function or cost. Categories of criteria include basic (e.g., maintainability and installation), management (e.g., vendor viability, costs, and required training), architecture (e.g., platform and framework), strategic (e.g., information technology goals and business goals), and functional, which is specific to COTS class and project context.

> *"Selections are often second-guessed. By following a systematic process for evaluating and selecting COTS products, such as CEP [Comparative Evaluation Process], you adequately capture and document the information necessary to defend the selection."*

Another feature is the contextual focus that is explicitly part of CEP. The context focus enables effective use of resources to ensure a deliberate and calculated evaluation. This is in contrast to generically exercising an alternative's functionality, which provides little or no insight into how well each alternative aligns with your needs.

Finally is the credibility feature. Confidence in the data gathered during the evaluation is gained by knowing and rating the credibility of the data source. The selection decision includes this credibility factor on how well the evaluator knows the data values.

Comparative Evaluation Process Activities

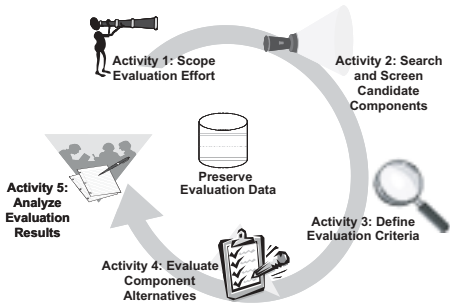
A systematic and repeatable process for evaluating and selecting COTS products provides the rationale necessary to support selection decisions made (e.g., pool of candidates, search criteria, minimum screening thresholds, alternatives to evaluate in depth, detailed evaluation criteria, and analysis). Selections are often second-guessed. By following a systematic process for evaluating and selecting COTS products, such as CEP, you adequately capture and document the information necessary to defend the selection.

CEP is made up of five top-level activities, which are explained below and depicted in Figure 1. Each activity has three to five sub-activities, which are explained in detail in the technical report [2].

Activity 1: Scope Evaluation Effort

This activity sets the expectations for the level of effort and schedule for the remaining activities within CEP. It provides the expected number of COTS products to search, screen, and evaluate. Feedback from future activities often requires redefining the scope for one or more of the activities. Feedback may

Figure 1: *CEP Diagram*



Credibility	Value	Description
Verified	10	Verified in-house using hands-on experience.
Demonstrated	7	Witnessed in a focused demonstration.
Observed	5	Seen but not studied.
Heard / Read About	3	Described by a user or vendor or seen in vendor or third-party literature.

Table 1: *Ordered List of Values Assigned to Credibility*

indicate that there were too many or too few possible candidate components located during the search or that the addition of criteria should change the scope. This activity allows you to plan resources while minimizing and identifying potential overruns.

Activity 2: Search and Screen Candidate Components

The search for candidates first requires that the initial search criteria and thresholds (the "must haves") be defined. The search criteria typically are based on required functionality and key constraints. Keep the criteria broad so that the search is not limited by too many constraints. Using the search criteria, perform a search for possible candidates from sources both internal and external to the project or organization. After locating the candidates, screen them by applying qualified minimum thresholds to the search criteria for each candidate. This allows the most promising candidates to be evaluated fully during Activity 4. Candidate screening is fundamental and cost effective because projects rarely have sufficient resources, budget, and schedule to evaluate every possible candidate.

Activity 3: Define Evaluation Criteria

This activity produces the detailed criteria necessary to support a repeatable and systematic evaluation. The definition of criteria refines, formalizes, and expands on the search criteria and addresses functional, architectural, management, strategic, performance, and financial characteristics of the candidates. Weights are established for all of the evaluation criteria with respect to each project's importance. The selection is based on criteria priority.

Activity 4: Evaluate Component Alternatives

The Evaluate Component Alternatives activity is conducted to assess how well the alternatives meet the defined criteria. Evaluation scenarios are developed to evaluate the alternatives within your particular context rather than generically exercising the alternative's functionality. Results are documented for analysis. While not all alternatives can or must be evaluated in the same manner, evaluation results are based on the available data. The available data may be from hands-on experience, witnessing vendor demonstrations, observing a user, and reading third-party literature or vendor's literature. Each type of data is given a rating value (Table 1). Credibility – rating the confidence in what the evaluator knows about an alternative – is then incorporated in the simple weighted average.

Activity 5: Analyze Evaluation Results

The evaluation produces data on how well each alternative meets the defined criteria. The analysis consists of activities to compare and contrast rankings of alternatives based on the priorities. Sensitivity analysis, using a decision-support method, is performed to determine the impact of criteria or groupings of criteria on the ranking of alternatives. More confident decisions may be made when the impact of the criteria is analyzed.

Decision Model

We developed an easy-to-use spreadsheet called the Decision Model that you can create yourself in a spreadsheet to hold the decision information (e.g., criteria, alternatives, priorities, ratings, and data charts). The Decision Model aids in decision making when comparing similar products using discriminating criteria.

- Software – Microsoft Excel.
- Decision Theory – Simple weighted average.
- Rows – Criteria.
- Columns – Alternatives.
- Cells – Criteria ratings for each alternative.

The following describes the Decision Model's basic features.

Decision Theory Model

The decision theory model behind the Decision Model is simple weighted averages. Simple weighted-average theory applies a weight to each criterion. The global weight is determined by multiplying weight, in percentages, by the weights of the criteria in the hierarchy. Assume the criteria hierarchy was as indicated in Table 2. Criteria 2.2 Vendor Viability has a local weight of 75 percent and is a subcriterion of 2.0 Management, which also has a local weight of 75 percent. To determine its global weights multiply 75 percent by 75 percent to equal 56.25 percent.

Weighting

Weights are applied to the evaluation criteria so that decisions can be made based on the results of the component evaluations. The weights are subjective and dependent on the particular project emphases. The decision-maker must provide a set of weights that are believed to be appropriate for the situation at hand. For the Decision Model, the weights in a level of the hierarchy must add up to 100 percent for normalization purposes. Additional averaging techniques such as dividing 100 points among the criteria or assigning them high, medium, and low values may be used and converted to a normalized scale.

Credibility

The purpose of credibility value scoring, as discussed above, is to include how well the evaluator knows the criteria value in the scoring equation. Often vendor-supplied information is not considered as valid as that verified through hands-on experience. The assignment of credibility values should reflect a level of confidence of the information contained in the criteria ratings value. To achieve this, Table 1 shows an example of an ordered list with the greatest confidence at the top of the list. This is reflected in the value assigned to each credibility scale item. The values are based on the experience of the evaluator. It is an attempt to quantify what is essentially qualitative.

Table 2: *Global Weight Calculation Example*

First Hierarchy Level Criteria (local weight)	Second Hierarchy Level Criteria (local weight)	Global Weight
1. Basic (25%)		
	1.1 Usability (50%)	12.5%
	1.2 Maintainability (50%)	12.5%
2. Management (75%)		
	2.1 Suggested Training (25%)	18.75%
	2.2 Vendor Viability (75%)	56.25%
TOTAL		**100%**

Calculating the Result

Using simple weighted average, the Decision Model calculates the results based on the criteria value and credibility ratings. Table 3 shows an example of the values entered into the simple weighted average.

The scoring uses a 10-point scale to normalize the data. For simplicity, the average is divided by 100 putting the result on a 100-point scale. Words used in the rating scale are converted to numbers using the Microsoft Excel function VLOOKUP. The set of values are named (Insert, Name, and Define) and referenced in the formula. In the example below, Excellent_Good_Fair_Value and Credibility_Value are defined names. The formula below would replace cell E3 in Table 3 if the named values are used and need to be converted to numbers. Alternatively, the numbers could be used directly.

```
E3=$B$3*VLOOKUP(C3,Excellent
_Good_Fair_Value,2,FALSE)*
VLOOKUP(D3,Credibility_Value,2,
FALSE)/100
```

How to Interpret Results

A special alternative in the Decision Model is the one named the Perfect. Its criteria rating and credibility ratings are set at the maximum values. For the bar chart showing the cumulative scores for each alternative, the Perfect is set at 100 percent. When the criteria are grouped, the Perfect allows comparison between the highest possible score and the alternative's score. For example, Figure 2 shows that Alt C has the highest ranking for the Functional Criteria Category at 18 percent and the Perfect score for the category is 35 percent. Clearly, none of the alternatives performed very well in this category. Strategic criteria was not pertinent to the evaluation and the category was dropped. The evaluator now knows the selected COTS product is not going to have all the desired functionality and may consider refining the criteria, negotiating the requirements, or finding another source to provide the missing functionality.

Sensitivity analysis is a method for determining confidence in the results. This enables decision making based on the impact that the criteria have on the selection of the COTS product. The sensitivity analysis may include operations such as reviewing the weights of the evaluation criteria, making adjustments to the weights, and observing the effect on the results. This activity may be performed multiple times depending upon what is observed or uncovered while doing the sensitivity analysis.

	A	B	C	D	E
1			ALTERNATIVE A		
2	**Criteria**	**Global Weight**	**Criteria Rating**	**Credibility Rating**	**Weighted Average**
3	2.2 Vendor Viability	56.25%	Excellent (convert to 10)	Verified (convert to 10)	(B3*C3*D3)/100

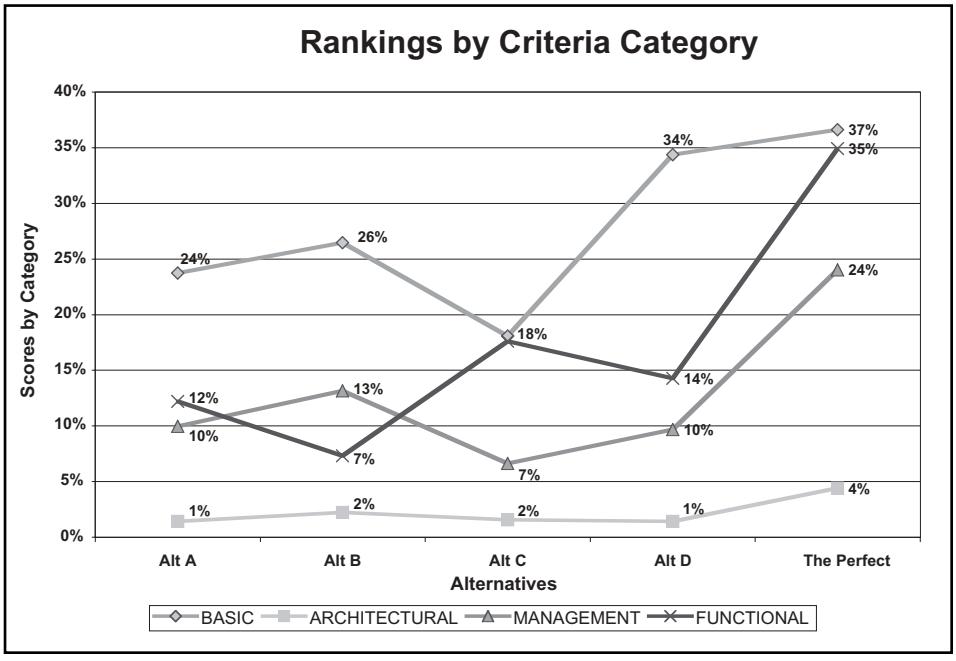
Table 3: *Calculations in the Decision Model*

Lessons Learned

Many lessons may be learned while applying CEP; all are of equal importance:

- Early and Effective Vendor Contact. Making contact and getting results to inquiries from vendors is a long and laborious effort. Do not underestimate effort and schedule for this activity. Staying on top of the communication flow helps prevent schedule slips. Smart vendors see the benefit of participating in an evaluation. The ones who have been most responsive are those who provided an explanation of how well their product performed against the criteria. We have made it a policy not to give vendors a copy of the evaluation report. It subjects us to too many unnecessary questions. Our process is focused on finding a product to fit the specific context of the evaluation, not a best in class, and vendors have a hard time understanding this.
- Look to Training Requirements for Information. Training requirements could be an indicator of the size or scope of installation and actual hands-on evaluation time. Typically, this ranges from none to one week. We attempted to install the products for one evaluation and finding it very difficult, discovered it required a week of administrator training.
- User Observation. Interviewing or observing a user may be a more practical and beneficial method of data collection over witnessing a vendor demonstration. The credibility factor can account for the difference in the source of ratings.
- Use of Evaluation Scenario and Data. To get the most out of a vendor demonstration, request that vendors provide a focused demonstration with materials from your evaluation scenario and data. Vendors typically have a set of features they want to show you, but they may not be the features in which you are interested.
- Subject Matter Experts. By obtaining the services of a subject matter expert to assist with the class of COTS products under evaluation, you can more efficiently identify possible candidates, define criteria, and develop an evaluation scenario and data.
- Estimation Data. Data used to estimate and scope the effort along with actual tracking data of the evaluation should be retained within the repository. It will provide historical data to be

Figure 2: *Ranking by Criteria Category Example*



used for estimating future evaluations.
- Demo Forum. Allowing stakeholders to learn and witness a demonstration of the final alternatives proves an excellent means for getting input and buy-in. More forums could be held to provide hands-on experience. A forum also may be appropriate for criteria identification.
- Advocate. Assigning an evaluation team member to be responsible for installing and learning a final alternative is an effective use of limited resources. The advocate becomes an expert in an assigned alternative and may rate the criteria with confidence.
- Software Installation. For those tools to be evaluated hands on, consider borrowing a vendor-owned laptop with the software already installed if installation problems become overwhelming. Capture this problem in the criteria ratings.
- Team Size. The ideal size for the evaluation team is between three and five. A smaller team may allow bias while bigger teams make communication complex and scheduling difficult.

Additional lessons may be learned while applying CEP to make confident COTS selection. Below are answers to the questions that began this article.
- Use a systematic and repeatable process such as CEP, which can be tailored and refined with each use to maximize its benefit.
- To ensure the best value for your desired features, translate your features into measurable criteria, assign priority to your criteria, rate your alternatives according to the criteria, and let simple weighted averages (or other decision-support method) provide the answer.
- Vendors are driven by current or potential profits. They can be cooperative and responsive when it is in their perceived interest to be so [3]. Never confuse selling with installing. Salespeople speak of the product's strengths but not the weaknesses. Factor in your data source (e.g., hands on, vendor demonstration, and vendor literature) when scoring the alternative criteria.
- A systematic and repeatable process for COTS evaluation and selection provides the rationale necessary to support decisions. The basis for the decision is available for review, which increases the confidence in the results.
- From a project management perspective, if the decision is important to the overall success of the project, then it should be given adequate resources. Those resources should be used efficiently and effectively, as is the case with CEP.
- Save all documentation (e.g., pool of candidates, search criteria, minimum acceptable threshold values, detailed evaluation criteria, alternatives to evaluate in depth, and analysis). The evaluation may need to be reviewed because of a new entry in the market or a new version of an existing tool. Maintain the evaluation data in a repository. It is helpful to see the artifacts from a completed evaluation when starting a new one to get ideas as candidate sources and criteria.

Table 4: *Comparison of the CEP and the CMMI DAR Process Area*

CEP	DAR Specific Practices (SP) and Pertinent Generic Practices (GP)
CEP	The purpose of Decision Analysis and Resolution is to analyze possible decisions using a formal evaluation process that evaluates identified alternatives against established criteria. GP 2.3 Provide adequate resources for performing the decision analysis and resolution process, developing the work products, and providing the services of the process. GP 3.1 Establish and maintain the description of a defined decision analysis and resolution process. GP 3.2 Collect work products, measures, measurement results, and improvement information derived from planning and performing the decision analysis and resolution process to support the future use and improvement of the organization's processes and process assets.
Activity 1: Scope Evaluation Effort	SP 1.4 Select the evaluation methods. GP 2.2 Establish and maintain the plan for performing the decision analysis and resolution process. GP 2.7 Identify and involve the relevant stakeholders of the decision analysis and resolution process as planned. GP 2.8 Monitor and control the decision analysis and resolution process against the plan for performing the process and take appropriate corrective action.
Activity 2: Search and Screen Candidate Component	SP 1.2 Establish and maintain the criteria for evaluating alternatives, and the relative ranking of these criteria. SP 1.3 Identify alternative solutions to address issues. SP 1.5 Evaluate alternative solutions using criteria and methods.
Activity 3: Define Evaluation Criteria	SP 1.2 Establish and maintain the criteria for evaluating alternatives, and the relative ranking of these criteria.
Activity 4: Evaluate Component Alternatives	SP 1.5 Evaluate alternative solutions using criteria and methods.
Activity 5: Analyze Evaluation Result	SP 1.6 Select solutions from the alternatives based on the evaluation criteria.

Comparative Evaluation Process and CMMI

Table 4 compares CEP with the DAR process area of the CMMI to show their relationship. The purpose of DAR is to make decisions using a structured approach that evaluates identified alternatives against established criteria.

Conclusion

In collaboration with our membership, the process has been successfully applied to select the following:
- Change Management Tools.
- Decision Analysis Tools.
- Knowledge Management Portals.
- Process Modeling and Simulation Applications.
- Voice Recognition Software.

In summary, a systematic approach to COTS evaluation was developed to help avoid common pitfalls associated with evaluations and trade studies. This approach assists evaluators with component selection. It is generally applicable to components and particularly to COTS software. It adapts decision-support methods to assist with implementation. The approach stresses the creation and maintenance of a repository for capturing evaluation data and lessons learned for future use. The Consortium has collaborated successfully with many members to select COTS products using CEP and is rapidly building a repository of completed evaluations.◆

References

1. Capability Maturity Model Integration (CMMI℠) for Systems Engineering/Software Engineering Integrated Product and Process Development. Ver. 1.1. Pittsburgh, Penn.: Software Engineering Institute, Dec. 2001. 530. Continuous Representation. Decision Analysis and Resolution (DAR).
2. Polen, Susan M., Louis C. Rose, and Barbara C. Phillips. Component Evaluation Process (SPC-98091-CMC, Version 01.00.02). Herndon, Va.: Software Productivity Consortium, 1999 <www.software.org/pub/darpa/darpa.html>.
3. Lessons Learned in Developing Commercial Off-the-Shelf (COTS) Intensive Software Systems. Federal Aviation Administration. Washington, D.C.: Software Engineering Resource Center, 2000.

About the Authors

Barbara Cavanaugh Phillips, certified Project Management Professional, is a senior member of the technical staff at the Software Productivity Consortium. Phillips is co-author of the Consortium's Comparative Evaluation Process (CEP) and has worked with the Consortium's membership to apply CEP. She has a bachelor's degree in American studies from George Washington University and a master's degree in information systems from the George Mason University. Phillips is a member of the Institute of Electrical and Electronics Engineers.

Software Productivity Consortium
SPC Building
2214 Rock Hill Road
Herndon, VA 20170
Phone: (703) 742-7309
Fax: (703) 742-7200
E-mail: phillips@software.org

Susan M. Polen is a senior member of the technical staff at the Software Productivity Consortium. Polen is co-author of the Consortium's Comparative Evaluation Process (CEP) and its supporting training courses, Web site, and Repository of Evaluations and has worked with a number of Consortium members to apply CEP. She has 13 years of software and system development experience including Motorola and Allied Signal. Polen has a bachelor's degree in computer science from Mary Washington College.

Software Productivity Consortium
SPC Building
2214 Rock Hill Road
Herndon, VA 20170
Phone: (703) 742-7178
Fax: (703) 742-7200
E-mail: polen@software.org

WEB SITES

Software Technology Support Center

www.stsc.hill.af.mil

The Software Technology Support Center (STSC) is an Air Force organization established to help other U.S. government organizations identify, evaluate, and adopt technologies to improve the quality and efficiency of their software products and their ability to predict delivery cost and schedule. The STSC Web site now provides mappings of the Capability Maturity Model for Software Version 1.1 to and from the Capability Maturity Model Integration℠ for Systems Engineering/Software Engineering/Integrated Product & Process Development Version 1.1.

Risk Management

www.acq.osd.mil/io/se/risk_management/index.htm

This is the Department of Defense (DoD) risk management Web site. The Systems Engineering group within the Interoperability organization formed a working group of representatives from the services and other DoD agencies involved in systems acquisition to assist in the evaluation of the DoD's approach to risk management. The group will continue to provide a forum that provides program managers with the latest tools and advice on managing risk.

INCOSE

www.incose.org

The International Council on Systems Engineering (INCOSE) was formed to develop, nurture, and enhance the interdisciplinary approach and means to enable the realization of successful systems. INCOSE works with industry, academia, and government in these ways:

- Provides a focal point for disseminating systems engineering knowledge.
- Promotes collaboration in systems engineering education and research.
- Assures the establishment of professional standards for integrity and in the practice of systems engineering.
- Encourages governmental and industrial support for research and educational programs to improve the systems engineering process and its practices.

Center for Software Engineering

http://sunset.usc.edu/index.html

Dr. Barry W. Boehm founded the Center for Software Engineering (CSE) in 1993. It provides an environment for research and teaching large-scale software design and development processes, generic and domain-specific software architectures, software engineering tools and environments, cooperative system design, and the economics of software engineering. One of CSE's main goals is to research and develop software technologies that can help reduce cost, customize designs, and improve design quality by doing concurrent software and systems engineering. It also aims for research topics that will facilitate the training and education of skilled software leaders.

The Open Group's Architectural Framework

www.opengroup.org

The Open Group is a vendor-neutral, international, member-driven standards organization. It focuses on the development of software standards that enable enterprise integration. The Open Group is a global network of information technology customers and vendors who are developing multi-vendor integration solutions through open standards, testing, certification, and branding. Members' benefits include the following:

- Advanced knowledge of technology and standards developments.
- The opportunity to participate in or lead the development of standards.
- Access to information on real-world implementations and proven practices.
- Better procurement practices supported by well-defined brands and standards.

Project Management Institute

www.pmi.org

The Project Management Institute (PMI) claims to be the world's leading not-for-profit project management professional association. PMI provides global leadership in the development of standards for the practice of the project management profession throughout the world.

Prerequisites for Success: Why Process Improvement Programs Fail

David Cottengim
Defense Finance and Accounting Service

Why do system development programs so often fail to meet their objectives? Why do efforts to implement software process improvement methodologies fail to yield promised results? Regardless of the improvement methodology chosen, there are fundamental prerequisites to success that must be present. Absent these prerequisites, any attempt to implement a structured approach to quality improvement will fail. It is not enough to infuse the tools of improvement into an organization. The fundamental nature of an organization must support the core business changes required to successfully implement an improvement program.

System development programs continue to fail at an alarming rate. Failure is defined as: a) coming in late, b) going over budget, or c) not delivering what was required [1]. Failure often comes only after millions of dollars in scarce resources have been invested in the doomed venture. In spite of all the existing research and lessons learned, system development programs remain recalcitrant. There is clearly a need to improve the quality of system development efforts [2].

The failure to accurately and completely identify the problem to be solved (system requirements) is a root cause of system development failures [3]. The following definitions will provide a base of understanding for this article:

- What is a system requirement? System requirements encompass a broad spectrum of capabilities and attributes that a business system must possess. To properly address the risks associated with requirement gathering, a sufficiently inclusive definition must be adopted. For the purposes of this discussion we use the following definition of system requirement:

  > Any function, capability, characteristic, constraint, or purpose the software system in question must directly or indirectly address or satisfy for any stakeholder [4].

- What is a system requirement risk? There are many formal definitions of risk. I choose to apply the following definition provided by the Project Management Institute:

  > A risk is an uncertain condition that, if it occurs, has a positive or negative effect on a project objective [5].

Where Do Requirements Originate?

System requirements should emanate from a business need [6]. Numerous methodologies exist for the sole purpose of requirement elicitation, gathering, and documentation. The impediments to successful requirement elicitation are numerous [7]. Multiple business variables converge on any effort to gather the requirement for a perceived business need.

"The seeds of system failure are often sown at this point in the requirements elicitation process. Many organizations lack the ability to consolidate and reconcile multiple stakeholder viewpoints ..."

Business problems must be evaluated in the context of strategic planning beyond the system solution being developed for any individual business need. Shortsighted solutions to immediate business needs might cause considerable long-term harm to the organization [8].

Stakeholder Viewpoints

Most software systems in today's business environment have stakeholders with divergent and conflicting points of view about the nature of the business problem, let alone how it should be solved. These varied points of view manifest in the requirement elicitation process and must be considered before a final system solution to a problem can be defined [9]. Figure 1 describes a basic requirement elicitation process.

Senior management should approach a business need from a strategic point of view. How does this problem fit into the organization's larger mission? Can a link be drawn between solving a particular business need and a larger organizational performance goal?

Middle management might also have a strategic slant to their point of view, but will generally also bring a near-term tactical point of view to what needs to be done about a business problem. Is this problem preventing the accomplishment of mission-critical functions? Will solving the perceived problem provide any benefits to their level of the organization? How will any proposed system solution impact their way of doing business?

Task level action officers bring yet another point of view to the requirement elicitation process. At the task execution level of an organization, there are very real concerns about how a new software application will impact day-to-day operations. Software systems addressing larger organizational objectives will often place additional workload on those at the task execution level. Job security, job satisfaction, advancement, need for training, skill set requirement, and retention of institutional knowledge are all areas of concern at the task execution level of the organization.

Defining the Final System Requirement

Documenting the different stakeholder points of view is not the end of the requirement definition process. The requirements generated by all the different points of view must be synthesized into a

single system requirement. The skills required to consolidate divergent requirements and to maneuver stakeholders to agreement are elusive.

The seeds of system failure are often sown at this point in the requirements elicitation process. Many organizations lack the ability to consolidate and reconcile multiple stakeholder viewpoints or to resolve conflicting requirements.

The pivotal transition for organizations wishing to reduce system requirement risks is the implementation of an underlying culture that facilitates the migration from current business practices and points of view to a new world view and mode of thought in support of the new system.

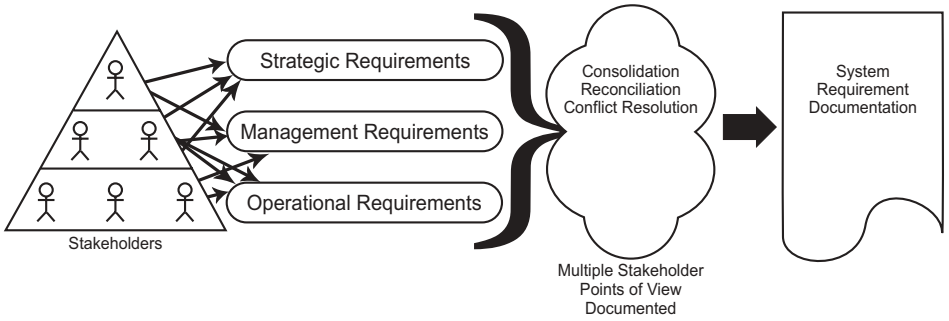


Figure 1: *Requirements Elicitation Process*

Requirements Elicitation Obstacles

Figure 2 presents the typical environmental obstacles all stakeholders must face as they try to define business requirements.

All stakeholders are impacted by four primary sources of system requirement risks. Each stakeholder group will build their own personalized amalgamation of these factors as they present what they perceive to be the business need and the correct system solution to address that need.

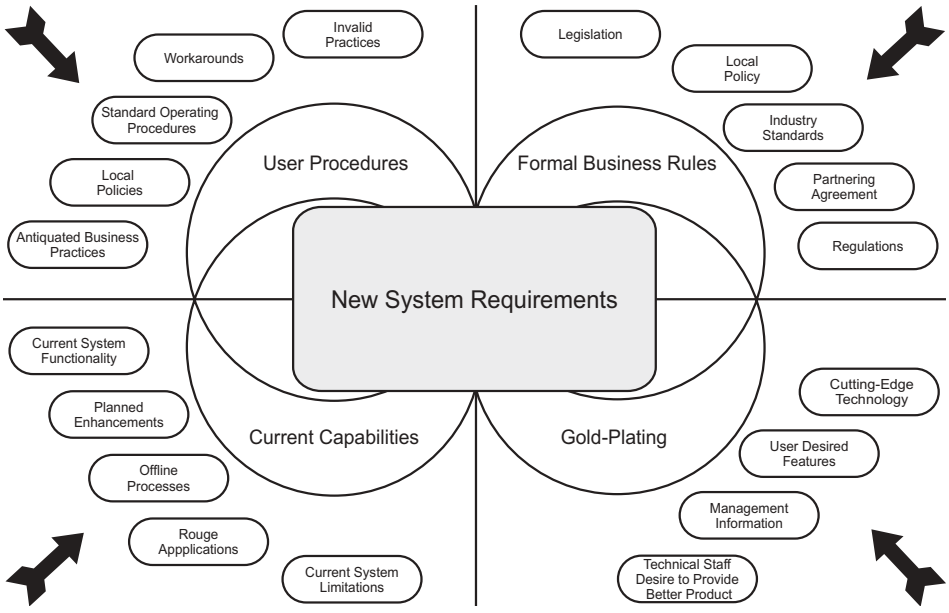
User Procedures

- Invalid Practices: Invalid day-to-day practices are often discovered during requirement elicitation exercises. Over time the end users will follow the path of least resistance to balance demands on their time and resources. This will often result in long-standing practices that are in direct violation of formal policy, guidelines, regulations, or legislation.
- Workarounds: Limitations of the current system solution almost always require the end users to develop workaround procedures to accomplish mission critical activities. These workarounds may not be documented or even formally acknowledged by the management of the business area. Workarounds may go on for years after the original need has vaporized.
- Standard Operating Procedures (SOPs): SOPs are often documented for a business activity. SOPs vary in degree of relevance and accuracy for current or future business needs. Depending on the degree of correlation between the SOPs, the real business need, and how things are really accomplished, use of the SOPs might not be a valid approach to defining system requirements for a new system.
- Local Policies: Tradition and local preference influence business practices. Local policies regarding sharing authority, delegation of duties, customized methods, and adherence to formal business rules will vary.
- Antiquated Business Practices: As technology and business practices evolve, it is not uncommon for personnel to hold on to old and familiar ways of doing business. Filtering outdated business practices out of the requirements for a new system solution can often be in conflict with the desire to make the system user friendly.

Current Capabilities

- Current System Functionality: The easiest trap to fall into is allowing current system capabilities to dictate future system capabilities [10]. The connection between current system capabilities and real business requirement is dubious at best. The program manager must maintain a focus on business needs.
- Planned Enhancements: Many current business applications have been in a production environment for decades. User requested enhancements are almost certainly planned and backlogged. Any effort to develop a new system will be saddled with a list of enhancements the old system was going to satisfy "any day now."
- Offline Processes: Completely mapping current business practices is the key to finding offline processes. End users usually developed offline processes to compensate for legacy system deficiencies. If an offline process cannot be linked to a current business need, it should not be a source of system requirements for a new system.
- Rogue Applications: Advances in desktop application capabilities have given managers and end users in all areas of operations the ability to build their own unique applications to meet their business needs [11]. The new system solution might need to interface with these rogue systems or absorb their functionality.
- Current System Limitations: Business practices are constrained by the limitations of legacy systems. Relying on

Figure 2: *Environmental Obstacles to Requirements Elicitation*



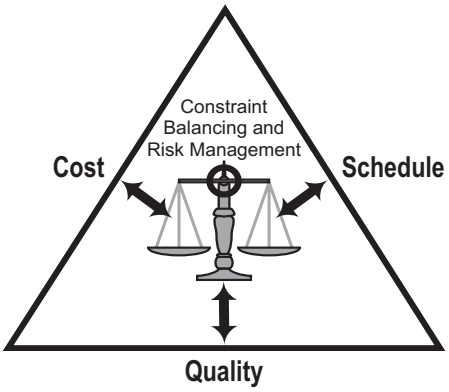


Figure 3: *Primary Constraints*

current business practices as the sole source of requirements is not the best technique for defining current business needs. Current business practices might have been overly tailored to accommodate material limitations in the current legacy systems.

Formal Business Rules

- Legislation: Deliberative bodies at all levels of government have attempted to address the performance of system acquisition programs. Numerous legislative requirements influence system solutions for government entities.
- Local Policy: Lack of standardized implementation methods for formal business rules and the resulting variation in business requirements will overwhelm attempts to develop a single system solution to a common business need. Peer-level organizations may lobby to have their way of doing business included as a mandatory feature of a new system solution to a business need.
- Industry Standards: Industry groups define standards for conducting business. These standards often influence requirements for system solutions to business needs. The integration and interdependence of business systems across organizational boundaries generate the need to stay current with applicable industry standards to assess their impact on planned system solutions.
- Partnering Agreements: Beyond the requirements of industry standards, there are usually requirements between organizations regarding how industry standards will be implemented or how other common business practices will be integrated. Standards only provide templates and guidelines. Additional work beyond what the standard provides is normally required before a working relationship between two organizations can be finalized.
- Regulations: Business rules for government-related activities often manifest in the form of published regulations. Regulations provide further guidance on implementation of policy or legislative requirements. Inconsistent interpretation of regulations is another source of variation in business processes for identical business needs.

Gold-Plating

Adding features or functionality to a system that are not required to satisfy the minimum operational requirements is referred to as "gold-plating" [12].

- Cutting-Edge Technology: Rapid advances in technological capabilities can entice businesses to introduce unnecessary capabilities into system requirement documentation. The desire to work with the latest and greatest technology is compelling.
- User Desired Features: Technologically savvy users will attempt to dictate the technical solution to their business need. Organizational level strategic goals and objectives must supercede user-level requests for specific technical solutions to business needs.
- Management Information: The quality and accuracy of management information is a top priority for system solutions to business needs. The presentation of the information can be a source of gold-plating. Users prefer information be presented in familiar formats. This might unnecessarily increase development costs. This is especially true when an activity is attempting to implement a commercial off-the-shelf package without making significant modifications to the core product.
- Technical Staff Desire to Provide Better Product: System engineers are often ingenious and will seek new and better technical solutions for business needs. Organizations must balance the three primary constraints of any project effort as depicted in Figure 3. As schedule and cost constraints become fixed, it is often necessary to compromise on quality and accept "good enough" system solutions [13].

Overcoming the Obstacles

What can an organization do to reduce system requirement risks and overcome these impediments to successful system development?

Contemporary Wisdom

Contemporary wisdom is to implement some version of the various process improvement programs as a method for reducing system requirement risks. Several software process improvement methodologies have risen to the forefront. These methodologies include the following:

1. The Capability Maturity Model® <www.sei.cmu.edu>.
2. International standards such as ISO/IEC 15504 (SPICE) <www.sei.cmu.edu/iso-15504> and ISO 9001 <www.iso.org/isoen/iso9000-14000/tour/ magical.html>.
3. Joint Application Development (JAD) [14].
4. Rapid Application Development (RAD) [15].
5. Quality Function Deployment (QFD) <www.qfdi.org>.
6. Six-Sigma <www.6-sigma.com>.

While each of these methodologies and techniques has shown positive results in specific implementations, none of them can claim to be the silver bullet of software process improvement [16].

Implementing these methodologies alone will not burrow down far enough into the core competencies of the organization. While these methodologies do present better techniques for reconciling and consolidating requirements and resolving conflicting requirements, they all share the same fundamental prerequisites for success.

Organizations should therefore first focus on the foundational prerequisites for success. Only after the prerequisite conditions have been secured can a methodology or technique significantly reduce system requirement risks.

Building the Foundation for Success

The environment surrounding system development will not become less complex. Organizations must adapt to the complexity of their environment. Implementing new requirements gathering methodologies without the attendant examination of the organization's underlying characteristics is too often the approach taken to address environmental complexity. The key to successfully adapting to increased environmental complexity is to focus management's attention on the characteristics of how the organization engages complexity.

Failure to create the prerequisite organizational character to foster the success of process improvement or risk management programs will cause the implementation of any methodology to be superficial and doom it to failure. The specifics of the process adopted are not nearly as critical as the philosophical change required to tran-

sition to the new paradigm [17].

Therefore management's focus should shift away from a particular methodology and toward the creation of an environment that meets the prerequisites for success under any of the possible methodologies. The success of any action to manage system requirement risks will depend on the dominating presence of several key prerequisites, as shown in Figure 4 [18].

Prerequisites for Success

1. Leadership: The cornerstone to any successful process improvement or risk management plan is leadership focused on clearly defined goals and objectives. This is very difficult to find even in small organizations and woefully lacking in large institutions. Leadership ambiguity will confuse and frustrate personnel and misdirect resources. Organizational leaders must remove obstacles and move the organization toward the selected goals and objectives [19]. Leadership vacuum will guarantee failure of any effort to materially reduce system requirement risks.
2. Commitment: Talk is cheap. Rhetoric is damaging. Management must be prepared to show firm commitment to risk management policies and process improvement methods in their own actions and in the actions they require from personnel. Commitment is needed from every member of the organization. The level of commitment among staff will vary but management must be committed to building teams of individuals that are fundamentally behind the improvement program. Team members must be supportive of the efforts to implement new initiatives [20]. Retention of key personnel who refuse to transition to the New World view required to support a system or methodology demonstrates a lack of management's commitment and will undermine risk reduction and process improvement efforts.
3. Honesty: Many organizations will not face the truth of their environment. Failures are often spun into successes. Yardsticks of success are shortened to declare victory when any objective evaluation would return a failure verdict. Organizations that cannot be honest about their shortcomings and failures will be powerless to take action for improvement. Stakeholders need to evaluate the results of their performance against honest objectives to determine their cause and interrelationship [21]. Pretending projects have gone well will breed cynicism. Rewarding failure de-motivates personnel. An environment of dishonesty cannot take the action required to foster improvement.
4. Training (a.k.a. education): Once a direction is chosen, the organization must be mobilized to support it. Training in the chosen methodology will show personnel the behavior that will be rewarded. Adopting behavior that supports the organization's risk management objectives is the critical path to successful process improvement. Education does not teach behavior. Organizations must take care to differentiate between education on a subject and training in a specific behavior [22]. The organization must then be honest when evaluating performance relative to the trained behavior.
5. Standardization: An organization cannot afford multiple methods for achieving the same outcome. This will be the stumbling block for many system implementations. Non-standard business methods will cripple system solutions to business needs. Stakeholders must insist the organization migrate toward standard processes and overcome resistance against transitioning to fewer optimized business processes [23]. Permitting multiple processes for accomplishing the same task will add complexity to system requirements that cannot be overcome by improved processes or risk management programs.
6. Professionalism: Requirement elicitation, cost estimating, system engineering, system testing, and project management have become formal professions. Each of these professions has a body of knowledge for practitioners to acquire and master. Professional certifications exist to help measure practitioner skill level. Adequately trained and skilled personnel should be hired and placed in these professional positions. A team of professionals committed to following proven methods in their work will produce consistent and predictable results that will demonstrate sustained improvement [24]. An honest assessment of skills will usually reveal significant skill-set deficiencies in personnel holding key leadership positions. Poor decisions by unskilled personnel will produce cascading impacts throughout a program. Hard work after the damage is done can rarely recover the opportunity costs of poor decisions.

Implications

If you choose to agree that the prerequisites for success must be present before any software process improvement or risk management program can succeed, then the implications of that belief are quite severe.

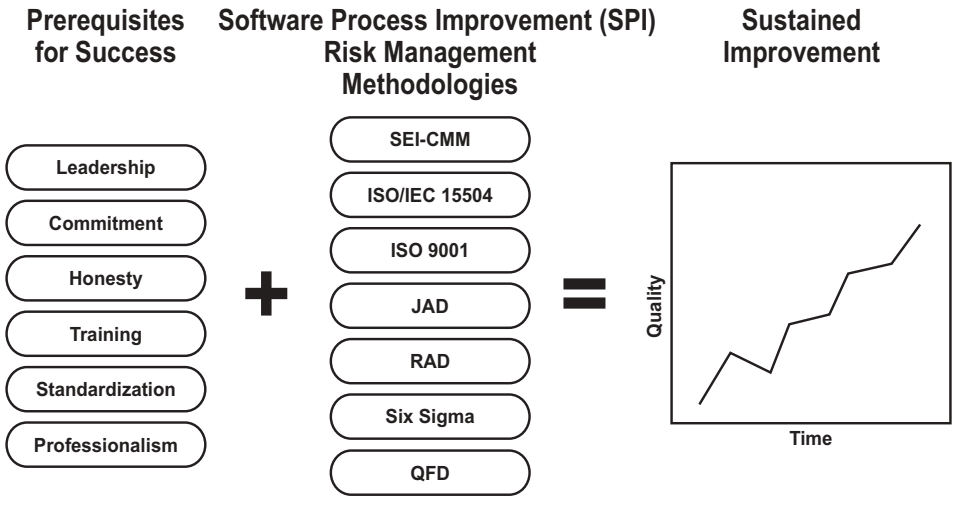
The mentality that suggests the definition of a good program manager is someone who can achieve success in an environment devoid of the prerequisites for success must be abandoned. That modality of leadership simply cannot overcome the obstacles confronting improvement programs.

If you agree that the prerequisites I have identified (see Figure 4) are required for success, then the conclusion follows naturally that some organizations are not ready to implement software improvement programs and should not be developing complex software systems.

Software improvement or risk reduction programs absent the prerequisites for success will continue to experience dismal performance in their attempts to develop software intensive systems [25]. This leaves you with the difficult task of introspection.

Before you invest the stakeholder

Figure 4: *Prerequisites for Success and SPI Risk Management Methodologies*



resources entrusted to you in any type of improvement program, a difficult decision awaits you. If you know your organization does not foster the environment required for success, then you have a fiduciary duty to not waste the resources you have been provided.

You must first take action to transform your organizational environment to one that provides the prerequisites for success, or you will simply be adding to the list of failed improvement efforts and cancelled system programs.◆

References

1. O'Connell, Fergus. How To Run Successful High-Tech Project-Based Organizations. Boston: Artech House, 1999. xvii.
2. Humphrey, Watts S. Managing the Software Process. New York: Addison-Wesley, 1989. 13.
3. Sommerville, Ian, and Pete Sawyer. Requirements Engineering: A Good Practice Guide. New York: Wiley, 1998. 64.
4. Young, Ralph R. Effective Requirements Practices. Boston: Addison-Wesley, 2001. 9.
5. Project Management Institute. A Guide to the Project Management Body of Knowledge. PMI Publishing, 2000. 127.
6. Wiley, Bill. Essential System Requirements: A Practical Guide to Event-Driven Methods. Reading, Mass.: Addison-Wesley, 2000. 15.
7. Gilb, Tom. Software Inspection. Addison-Wesley, 1993. 253.
8. Weinberg, Gerald M. Quality Software Management, Vol. 1, Systems Thinking. New York: Dorset House, 1992. 155.
9. Metzger, Philip, and John Boddie. Managing A Programming Project. New Jersey: Prentice Hall, 1996. 19.
10. Kotonya, Gerald, and Ian Sommerville. Requirements Engineering Processes and Techniques. New York, 1998. 171.
11. Jones, Capers. Assessment and Control of Software Risks. N.J.: Yourdon, 1994. 37.
12. Wiegers, Karl E. Software Requirements. Microsoft Press, 1999. 13.
13. Yourdon, Edward. Death March. N.J.: Prentice Hall, 1997. 147.
14. Wood, Jane, and Denise Silver. Joint Application Development. New York: John Wiley & Sons Inc., 1995.
15. McConnel, Steve. Rapid Development. Microsoft Press, 1996. 2.
16. Kemerer, Chris F. Software Project Management Readings and Cases. Boston: McGraw-Hill, 1997. 591, 600.
17. Horch, John W. Practical Guide to Software Quality Management. Boston: Artech House, 1996. 192.
18. Schulmeyer, Gordon G., ed., and James I. McManus. Handbook of Software Quality Assurance, 3rd ed. N.J.: Prentice Hall, 1999. 61.
19. Smith, Perry M. Rules & Tools for Leaders: How to Run an Organization Successfully. New York: Avery, 1998. 33.
20. IEE Computer Society. Software Engineering Project Management. IEE Computer Society, 1997. 380.
21. Aurelius, Marcus. The Meditations Of Marcus Aurelius, Book 12. Trans. A.S.L. Farquharson. Knopf, 1944. Chapters 10 and 29.
22. Beer, Michael. The Critical Path to Corporate Renewal. Harvard Business School, 1990.
23. Harry, Mikel, and Richard Schroeder. Six Sigma. New York: Currency, 2000. 134.
24. Humphrey, Watts S. A Discipline For Software Engineering. Mass.: Addison-Wesley, 1995. 474.
25. Hall, Elaine M. Managing Risk, Methods For Software Systems Development. Addison-Wesley, 1998. 152.

About the Author

David Cottengim is a financial analyst at the Defense Finance and Accounting Service, Indianapolis. He has more than 10 years experience in system development for Department of Defense activities. He is certified as a Project Management Professional, Certified Software Test Engineer, and Certified Government Financial Manager. He completed his undergraduate education in finance and economics and his graduate education in finance and management information systems at the Indiana University Kelly School of Business.

Defense Finance and Accounting Service - Indianapolis
Building #1, Column 230F
8899 East 56th Street
Indianapolis, IN 46249
Phone: (317) 510-3121
Fax: (317) 510-3174
E-mail: david.cottengim@dfas.mil

Risqué Requirements

I thought the theme for this issue was risqué requirements? By definition, that would be requirements offensive to established standards of decency. I envisioned the cover of the issue adorned by a young Tom Cruise skating across a hardwood floor in a pair of white socks, oxford shirt, and his famous tighty-whites. But that was "Risky Business" – which is what you have if requirements are misconstrued.

Requirements are important in developing effective software, so what's new? Twenty years ago we knew poor requirements were a major cause of software troubles. What have we done in those years? Studied, analyzed, decomposed, recomposed, processed, and defined the requirements of the business of requirements.

We made lists. In this issue alone we have nine lists for requirement risks, eight strategies to mitigate requirements risk, eight good requirements' characteristics, 16 recommended requirements gathering techniques, eight critical attributes of requirements, and 19 sources of system and requirement risks. We also brought out the alphabet soup of remedies – JAD, RAD, UML, DOORS, QFD, SPICE, CMM, Six Sigma, etc.

With respect to the work performed on requirements, I think the industry has missed the main point – to understand customer needs. The key noun is "customer" and the key verb "understand."

To understand a customer, we communicate. You think software engineers would understand communication, as it is the wellspring of our commerce. It's basic; you transmit, and you receive. We spend the majority of our vigor amplifying transmitter power while our receivers run on vacuum tubes, or worse, are vacuums – absent of matter. Yet, reception is vital in acquiring accurate and effective requirements and entails good listening skills. Not just to hear but to listen, pay attention, heed, be au fait with, and comprehend.

I think engineers have problems listening. Don't believe me? I have $20 for the first reader to find the word "listen" or "listening" in this issue on requirements, outside of this article.

Engineers would rather decipher the words to the Kingsmen's 1963 classic party song "Louie Louie" than decipher customer requirements. It's time we go beyond gathering requirements and focus on comprehending requirements.

> *"… for it remains true that those things which make us human [engineers] are, curiously enough, always close at hand. Resolve then, that on this very ground, with small flags waving and tinny blast on tiny trumpets, we shall meet the enemy, and not only may he be ours, he may be us."*[1]

For those in management, "We have met the enemy ... and he is us."

There are several reasons for this inadequacy. First, engineers are problem-solving mavens. We have the answers; why would we have to listen?

Let me demonstrate. Two buckets both two feet high and four feet in diameter containing equal mass of water are put outside on a Utah Olympic day (far below 0 degrees Celsius). One bucket's water temperature is 100 C and the other's is 50 C. Which one freezes first?

Are you solving the problem? Do you have the answer? You should be asking at least one important question. What are the buckets made of?

If the buckets are zinc-coated iron or steel, the 50 C bucket will freeze first. It starts at a cooler temperature and heat transfer is dominantly through the bucket's sides. If the buckets are wooden, the 100 C bucket will likely freeze first. Greater evaporation of the hot water carries off more water mass so that less water needs to be cooled. Also, evaporation carries off the hottest molecules, lowering the average kinetic energy of those remaining. Evaporation makes up for the temperature difference given the volume and surface area of the water and insulation of the wood.

Second, engineers often listen for content void of context and intent. John F. Kennedy's famous statement, "Ich bin ein Berliner," was grammatically correct but ambiguous and uncommon. It is like saying "I am a Hamburger," instead of "I'm from Hamburg." "Berliner" denotes a person from Berlin and "Pfannkuchen" denotes a jelly donut. Outside of Berlin, a "Pfannkuchen" is a pancake, so the term "Berliner Pfannkuchen" was used to denote the jelly donut in Berlin, which usually gets shortened to "Berliner." Berlin natives understood Kennedy because they understood context and intent. Engineers, on the other hand, thought he was a gooey pastry.

Third, engineers view requirements as constraints to creativity. *The Cat in the Hat* was Ted Geisel's response to John Hersey's revolutionary article "Why Can't Johnny Read?" Geisel used a predetermined list of 223 words to create the classic alternative to Dick and Jane. It's rumored that Geisel wrote *Green Eggs and Ham* on a bet from his publisher, Bennet Cerf, to write a book using only 50 different words. Dr. Seuss did not see constraints; he saw challenges.

Finally, engineers have tin ears for prosody, discourse, and rhetoric. Take the expression "I could care less." My colleagues suggest this expression of disdain is illogical and should be "I couldn't care less." They argue that if you could care less than you do, you really do care, the opposite of what you are trying to say.

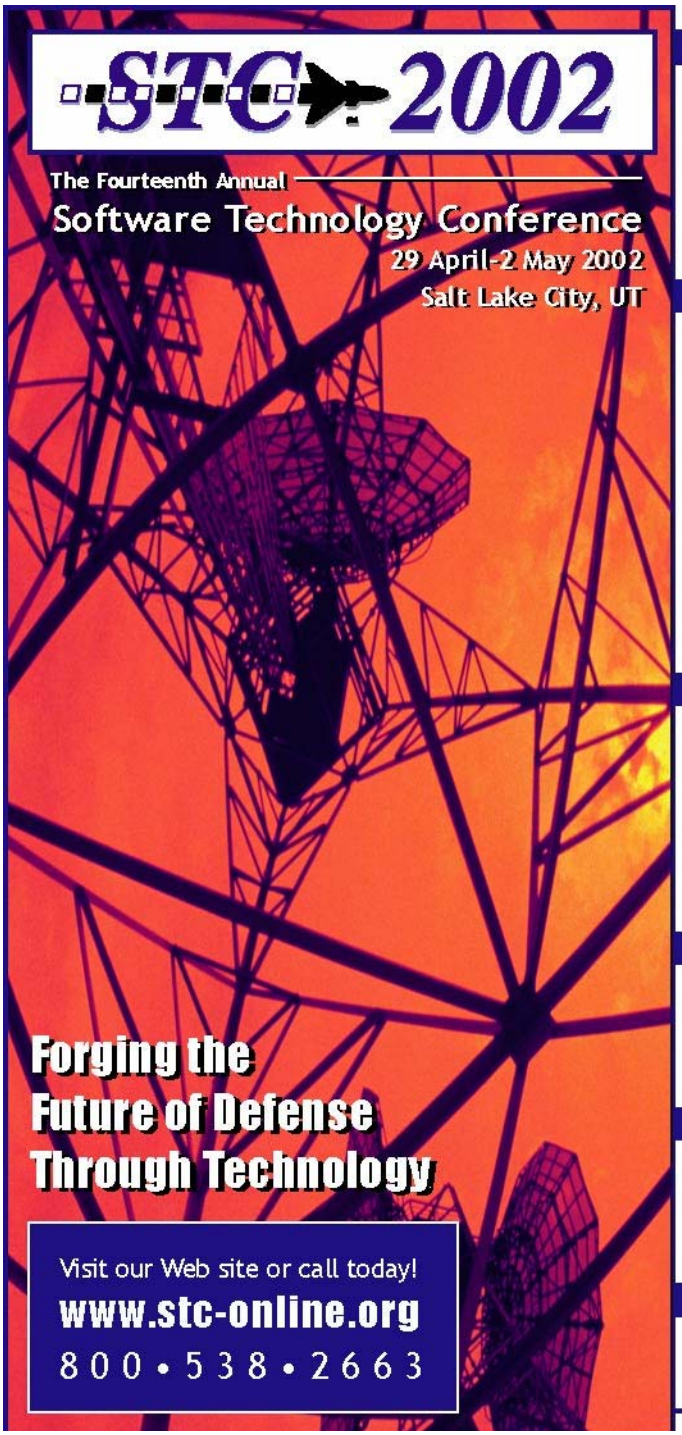
Lighten up Spock! Stop focusing on the logic and listen to the stress and intonation. The original expression is pronounced – I could CARE LESS. It's not illogical; it's sarcastic. The point of sarcasm is to make an assertion that is manifestly false or accompanied by ostentatious intonation, to deliberately imply its opposite. I know; you could care less.

As a précis, I sense engineers have a hard time listening because the principles, techniques, and skills associated with listening are considered soft science. Engineering is based on hard science. It would be easier to turn Luke Skywalker to the dark side than it would to turn an engineer on to soft science. How ironic, we build software but we snub soft science.

Looking at the software industry's record, our requirements are risqué – offensive to established standards of listening. Are you listening?

— Gary Petersen,
Shim Enterprise, Inc.

1. From the foreword to *The Pogo Papers*, Copyright 1952-53.



Opening General Session

Congressman James V. Hansen (R)
1st District of Utah (Invited)

Lloyd K. Mosemann, II
Senior Vice President
for Corporate Development
SAIC

Speaker Luncheons

Stephen L. Squires
Vice President and Chief Science Officer
Hewlett-Packard Co.

Dr. Margaret E. Myers
Acting Deputy Assistant
Secretary of Defense,
Deputy Chief Information Officer
Department of Defense

Chris Inglis
Deputy Director for Analysis and Production
National Security Agency

Industry Plenary Speakers

Kevin Fitzgerald
Senior Vice President Oracle Government,
Education and Health Care
Oracle Corp.

Grady Booch
Chief Scientist
Rational Software Corp.

Closing General Session

Tom Talleur
Managing Director in Forensic and Litigation
Services
KPMG

Tutorial Tracks

CMMI[SM] • Common and Open Systems
Enterprise Software • Process Improvement
Project Management • Quality Assurance
Requirements • XML

Sponsored Tracks

CIPO • CRSIP • DISA • Earned Value
IEEE • INCOSE • Joint Strike Fighter
OSD • STSC

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

CrossTalk / TISE
7278 4th Street
Bldg. 100
Hill AFB, UT 84056-5205

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