

CROSSTALK

October 2004 *The Journal of Defense Software Engineering* Vol. 17 No. 10



- 4 2004 U. S. Government's Top 5 Programs**
The annual Top 5 contest to award quality software development has a new name, and its scope has expanded to include total program performance.

Project Management

- 5 Software Project Management Practices: Failure Versus Success**
This author's analysis of successful large software projects versus those that ran late, were over budget, or cancelled reveals six common problems associated with project management rather than with technical personnel.
by Capers Jones

- 10 Catastrophe Disentanglement: Getting Software Projects Back on Track**
Most software calamities were troubled projects that went on for too long. This article proposes a 10-step process to effectively deal with an out-of-control project and get it back on track.
by E.M. Bennatan

- 15 Understanding Causal Systems**
This article describes a model and a supporting set of terms that facilitate reasoning about and planning for causal systems and designing process experiments, all of which is based on practical experience.
by David N. Card

Software Engineering Technology

- 19 Requirements Engineering So Things Don't Get Ugly**
Even if you know exactly what you want, requirements engineering is a tough task that requires understanding different points of view. This author discusses how processes in the Capability Maturity Model Integration provide a good foundation to accomplish this.
by Deb Jacobs

Open Forum

- 26 Independent Estimates at Completion – Another Method**
This article reviews the most frequently used Earned Value Management formulas for calculating the Independent Estimate at Completion (IEAC), and proposes an alternative method of calculating IEAC that shows promise.
by Walt Lipke



Departments

- 3 From the Publisher**
- 9 Coming Events**
- 18 Call for Articles**
- 25 Web Sites**
- 31 BACKTALK
CROSSTALK Archives**

CROSS TALK

**OC-ALC/ MAS
CO-SPONSOR** Kevin Stamey

**OO-ALC/MAS
CO-SPONSOR** Randy Hill

**WR-ALC/MAS
CO-SPONSOR** Tom Christian

PUBLISHER Tracy Stauder

ASSOCIATE PUBLISHER Elizabeth Starrett

MANAGING EDITOR Pamela Palmer

ASSOCIATE EDITOR Chelene Fortier-Lozancich

ARTICLE COORDINATOR Nicole Kentta

**CREATIVE SERVICES
COORDINATOR** Janna Kay Jensen

PHONE (801) 775-5555

FAX (801) 777-8069

E-MAIL crosstalk.staff@hill.af.mil

CROSSTALK ONLINE www.stsc.hill.af.mil/
crosstalk

Oklahoma City-Air Logistics Center (OC-ALC), Ogdan-Air Logistics Center (OO-ALC), and Warner Robins-Air Logistics Center (WR-ALC) MAS Software Divisions are the official co-sponsors of CROSS TALK, The Journal of Defense Software Engineering. The MAS Software Divisions and the Software Technology Support Center (STSC) are working jointly to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

The **STSC** is the publisher of CROSS TALK, providing both editorial oversight and technical review of the journal.



Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 18.

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

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSS TALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSS TALK does not pay for submissions. Articles published in CROSS TALK remain the property of the authors and may be submitted to other publications.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSS TALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSS TALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, or the STSC. All product names referenced in this issue are trademarks of their companies.

Coming Events: Please submit conferences, seminars, symposiums, etc. that are of interest to our readers at least 90 days before registration. Mail or e-mail announcements to us.

CrossTalk Online Services: See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-7026, or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.



CROSSTALK Welcomes New Sponsors



I am pleased to announce that the three U.S. Air Force Air Logistics Centers' (ALC) Software Divisions have joined together to become CROSSTALK's new co-sponsors. The three maintenance directorate divisions are commonly referred to by their office symbol, MAS, and are located at Ogden ALC, Hill Air Force Base, Utah; Oklahoma City ALC, Tinker Air Force Base, Oklahoma; and Warner Robins ALC, Robins Air Force Base, Georgia. Well known for high-quality software development and sustainment capabilities, the divisions are currently under the chief leadership of Randy Hill, Kevin Stamey, and Tom Christian, respectively. You will see little change to CROSSTALK as the journal's mission remains the same:

To encourage the engineering development of software in order to improve the reliability, sustainability, and responsiveness of our war fighting capability and to inform and educate readers on up-to-date policy decisions and new software engineering technologies.

Also, the Air Force Software Technology Support Center will continue in its role as the publisher as it has since CROSSTALK's inception in 1988.

We proudly begin this issue by announcing the fourth annual U.S. Government's Top 5 Programs contest, formerly called the Top 5 Quality Software Projects. The National Defense Industrial Association will facilitate the award process this year. You can submit your 2004 nomination at <www.ndia.org>.

This month we highlight project management and begin with an article from a longtime CROSSTALK supporter, Capers Jones. In his special report to CROSSTALK, *Software Project Management Practices: Failure Versus Success*, Jones looked at 250 large software projects. He found six common problem areas: project planning, cost estimating, measurements, milestone tracking, change control, and quality control. Learn how project managers focusing on these six areas can increase their project's chance of success.

Next, *Catastrophe Disentanglement: Getting Software Projects Back on Track* by E.M. Bennatan is featured in our theme section. This article describes how a project catastrophe can be determined through budget, schedule, or quality aspects and presents a 10-step process to aid a project manager and his or her team in turning around their project before it's too late.

As organizations continue to journey to high process maturity levels, teams may find themselves faced with applying causal analysis to identify defects or problems and their associated symptoms, causes, and corrective actions. In our final theme article *Understanding Causal Systems* by David N. Card, the basic concepts and terminology of causal systems are defined along with a model to facilitate reasoning.

In our first supporting article, *Requirements Engineering So Things Don't Get Ugly* by Deb Jacobs, we are reminded of the importance of customers and development teams working together to communicate, understand, and define effective requirements throughout a project lifecycle. This author discusses the basics of requirements engineering and defines key steps that a project manager can take to ensure requirements are defined, analyzed, and managed properly.

Our issue wraps up with *Independent Estimates at Completion – Another Method* by Oklahoma's MAS Deputy Chief Walt Lipke. Used to predict the final cost of a project, an Independent Estimate at Completion (IEAC) is a method often used by cost analysts and project managers. In this article, Lipke reviews several common calculations for IEAC and offers insights into why optimistic and questionable results may occur and thus proposes alternative calculations.

As we begin a new fiscal year at CROSSTALK, I welcome our new co-sponsors and look forward to their insights into the many lessons learned gained by their individual software divisions. Through this new partnership, we strengthen our commitment to disseminate information aimed at helping the defense software community acquire, develop, and sustain software better.

Tracy L. Stauder
Publisher



ACQUISITION,
TECHNOLOGY
AND LOGISTICS

OFFICE OF THE UNDER SECRETARY OF DEFENSE

3000 DEFENSE PENTAGON
WASHINGTON, DC 20301-3000

MEMORANDUM FOR ALL GOVERNMENT PROGRAM OFFICES

SUBJECT: 2004 U.S. GOVERNMENT'S TOP 5 PROGRAMS AWARDS

As the Department of Defense's Executive Agent for Systems Engineering and sponsor for activities aimed at improving acquisition, I am pleased to announce the search for the 2004 U.S. Government's Top 5 Programs, formerly the Top 5 Software Quality Projects.

Many organizations are employing processes and practices that result in the successful delivery of programs with significant software content to the United States government. Looking at past winners of this award, it is apparent that successful programs have used well-defined and proven processes and practices to develop, manage, and integrate software into deliverable systems. Beginning in 2004, this award intends to identify successful programs and highlight their efforts.

One significant change to the award structure this year, beyond extending the criteria from just software quality performance to total program performance, is the recognition of both the U.S. government project office and the industry prime contractor that participated in the system development, in recognition that successful programs are indeed government/industry team efforts. To facilitate the joint award process, we now have a co-sponsor of this prestigious award, the National Defense Industrial Association Systems Engineering Division.

CrossTalk will announce the Top 5 government and industry winners in the May 2005 issue, and winners will receive their awards at the 2005 Systems & Software Technology Conference. The winning projects will then be highlighted in a series of articles in *CrossTalk's* July 2005 issue.

Nomination forms and additional information on the U. S. Government's Top 5 Programs awards can be found at:

<http://www.ndia.org>, click on Divisions, then Systems Engineering.

Access to articles discussing previous winners can be found at:

<http://www.stsc.hill.af.mil/top5projects>

David R. Castellano
Deputy Director, Systems Engineering
Defense Systems
(Assessments and Support)



Software Project Management Practices: Failure Versus Success[©]

Capers Jones

Software Productivity Research LLC

An analysis of approximately 250 large software projects between 1995 and 2004 shows an interesting pattern. When comparing large projects that successfully achieved their cost and schedule estimates against those that ran late, were over budget, or were cancelled without completion, six common problems were observed: poor project planning, poor cost estimating, poor measurements, poor milestone tracking, poor change control, and poor quality control. By contrast, successful software projects tended to be better than average in all six of these areas. Perhaps the most interesting aspect of these six problem areas is that all are associated with project management rather than with technical personnel. Two working hypotheses emerged: 1) poor quality control is the largest contributor to cost and schedule overruns, and 2) poor project management is the most likely cause of inadequate quality control.

This article is derived from analysis of about 250 large software projects at or above 10,000 function points in size that were examined by the author's company between 1995 and 2004. (Note that 10,000 function points are roughly equivalent to 1,250,000 statements in the C programming language.)

It is difficult during analysis to pick out successful or unsuccessful methods from projects that are more or less average. However when polar opposites are examined, some very interesting differences stand out. The phrase *polar opposites* refers to projects at opposite ends of the spectrum in terms of achieving cost, schedule, and quality targets. When projects that were late by more than 35 percent, or overran their budgets by more than 35 percent, or experienced serious quality problems after delivery are compared to projects without such issues, some interesting patterns can be seen.

Of the 250 projects analyzed, about 25 were deemed successful in that they achieved their schedule, cost, and quality objectives. About 50 had delays or overruns below 35 percent, while about 175 experienced major delays and overruns, or were terminated without completion. The projects included systems software, information systems, outsourced projects, and defense applications. This distribution of results shows that large system development is a very hazardous undertaking. Indeed, some of the failing projects were examined by the author while working as an expert witness in breach-of-contract litigation involving the failed projects.

These large applications included both systems software and information systems. Both corporations and government

agencies were included. In terms of development methods, both waterfall development cycles and spiral development were included. The newer *agile* methods were not included because such methods are seldom if ever utilized on applications larger than about 1,000 function points.

Table 1 shows six major factors noted at opposite ends of the spectrum in terms of failure versus success as they were revealed in the study analysis.

The author and his colleagues were commissioned by clients to examine the software development practices, tools utilized, quality, and productivity results of various projects. Thus, this article may be biased toward the topics examined. We were not commissioned to examine other kinds of issues such as poor training, staff inexperience, or poor personnel practices. There are, of course, many other influential factors besides these six in this report. Indeed, several prior books by the author cited more than 100 factors [1, 2]. But these six key factors occur so frequently that they stand out from factors that occur only now and then. For additional studies on recent project failures other than the author's, see [3, 4, 5, 6].

Before dealing with the patterns observed on the successful and failing projects, it is desirable to discuss some of the differences between project planning

and project estimating since these are the key factors associated with both success and failure.

The phrase *project management tools* has been applied to a large family of tools whose primary purpose is sophisticated scheduling for projects with hundreds or even thousands of overlapping and partially interdependent tasks. These tools are able to drop down to very detailed task levels, and can even handle the schedules of individual workers. A few examples of tools within the project management class include Artemis Views, Microsoft Project, Primavera, and Project Manager's Workbench.

However, the family of project management tools is general purpose in nature and does not include specialized software sizing and estimating capabilities as do the software cost estimating tools. Neither do these general project management tools deal with quality issues such as defect removal efficiency. Project management tools are useful, but software requires additional capabilities to be under full management control.

The software cost estimation industry and the project management tool industry originated as separate businesses with project management tools appearing in the 1960s, around 10 years before software cost estimating tools. Although the two were originally separate businesses,

Table 1: *Opposing Major Factors in Study Analysis*

Successful Projects	Failing Projects
Effective project planning	Inadequate project planning
Effective project cost estimating	Inadequate cost estimating
Effective project measurements	Inadequate measurements
Effective project milestone tracking	Inadequate milestone tracking
Effective project change management	Ineffective change control
Effective project quality control	Inadequate quality control

© 2001-2004 Capers Jones.

they are now starting to join together technically.

Examples of specialized software cost estimating tools include Before You Leap, CHECKPOINT, Constructive Cost Model (COCOMO) II, CostXpert, KnowledgePlan, Parametric Review of Information for Costing and Evaluation – Software (PRICE-S), Software Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM), and Software Life Cycle Management (SLIM).

Project management tools are an automated form of several techniques developed by the Navy for controlling large and complex weapons systems. For example, the *program evaluation and review technique* (PERT) originated in the 1950s for handling complex military projects such as building warships. Other capabilities of project management tools include critical path analysis, resource leveling, and production of Gantt or timeline charts. There are many commercial project management tools available such as Artemis Views, Microsoft Project, Primavera, Project Manager's Workbench, and more.

Project management tools did not originate for software, but rather for handling very complex scheduling situations where hundreds or even thousands of tasks need to be determined and sequenced, and where dependencies such as the completion of a task might affect the start of subsequent tasks.

Project management tools have no built-in expertise regarding software as do the commercial software cost estimating tools. For example, if you wish to explore the quality and cost impact of an object-oriented programming language such as Smalltalk, a standard project management tool is not the right choice.

By contrast, many software cost estimating tools have built-in tables of programming languages and will automatically adjust the estimate based on which language is selected for the application.

Since software cost estimating tools originated about 10 years after commercial project management tools, the developers of software cost estimating tools seldom tried to replicate project management functions such as construction of detailed PERT diagrams or critical path analysis. Instead, the cost estimation tools would export data to a project management tool. Thus, interfaces between software cost estimating tools and project management tools are now standard features in the commercial estimation market.

Let us now turn to applying project planning and project estimating tools to large software applications.

Successful and Unsuccessful Project Planning

The phrase *project planning* encompasses creating work breakdown structures, and then apportioning tasks to staff members over time. Project planning includes creation of various timelines and critical paths including Gantt charts, PERT charts, or the like.

Effective project planning for large projects in large corporations involves both planning specialists and automated planning tools. Successful planning for large software projects circa 2004 involves the following:

- Using automated planning tools such as Artemis Views or Microsoft Project.
- Developing complete work breakdown structures.
- Conducting critical path analysis of project development activities.
- Considering staff hiring and turnover during the project.
- Considering subcontractors and international teams.
- Factoring in time for requirements gathering and analysis.
- Factoring in time for handling changing requirements.
- Factoring in time for a full suite of quality control activities.
- Considering multiple releases if requirements growth is significant.

Successful projects do planning very well indeed. Delayed or cancelled projects, however, almost always have planning failures. The most common planning failures include (1) not dealing effectively with changing requirements; (2) not anticipating staff hiring and turnover during the project; (3) not allotting time for detailed requirements analysis; and (4) not allotting sufficient time for inspections, testing, and defect repairs.

Successful project planning tends to be highly automated. There are at least 50 commercial project-planning tools on the market, and successful projects all use at least one of these. Not only are the initial plans automated, but also any changes in requirements scope or external events will trigger updated plans to match the new assumptions. Such updates cannot be easily accomplished via manual methods; planning tools are a necessity for large software projects.

Successful and Unsuccessful Project Cost Estimating

Software cost estimating for large software projects is far too complex to be performed manually. This observation is supported by the presence of at least 75 com-

mercial software cost estimating tools, including such well-known tools as COCOMO II, CostXpert, KnowledgePlan, PRICE-S, SEER-SEM, SLIM, and the like [7]. Successful projects all use at least one such tool, and usage of two or more is not uncommon. Estimates produced by trained estimating specialists are also noted on many successful large projects, but not on failing projects. Successful cost estimating for large systems involves using the following:

- Software estimating tools (COCOMO II, CostXpert, KnowledgePLAN, PRICE-S, SEER-SEM, SLIM, etc.).
- Formal sizing approaches for major deliverables based on function points.
- Comparison of estimates to historical data from similar projects.
- Availability of trained estimating specialists or project managers.
- Inclusion of new and changing requirements in the estimate.
- Inclusion of quality estimation as well as schedule and cost estimation.

By contrast, large failing projects may not utilize any of the commercial software estimating tools. However, manual estimates are never sufficient for projects in the 10,000-function point range.

Failing projects tend to understate the size of the work to be accomplished due to inadequate sizing approaches. Failing projects also omit quality estimates, which are a major omission since excessive defect levels slow down testing to a standstill. Overestimating productivity rates or assuming that productivity on a large system will be equal to productivity on small projects are other common reasons for cost and schedule overruns. The main problem with estimates for projects in the 10,000-function point size range is that they err on the side of excessive optimism.

Project planning tools and project estimating tools overlap in functionality, and are usually marketed separately. Normally, the project planning and cost estimating tools pass information back and forth. The software cost estimating tool would be used for overall project sizing, resource estimating, and quality estimating. The project-planning tool would be used for critical path analysis, detailed scheduling, and for work breakdown structures.

Successful and Unsuccessful Project Measurements

Successful large projects are most often found in companies that have software measurement programs for capturing productivity and quality historical data [8,

9]. Thus any new project can be compared against similar projects to judge the validity of schedules, costs, quality, and other important factors. The most useful measurements for projects in the 10,000-function point domain include measures of the following:

- Accumulated effort.
- Accumulated costs.
- Development productivity.
- Volume and rate of requirements changes.
- Defects by origin.
- Defect removal efficiency.

Measures of effort should be granular enough to support work breakdown structures. Cost measures should be complete and include development costs, contract costs, and costs associated with purchasing or leasing packages. There is one area of ambiguity even for top companies and successful projects: The overhead or burden rates established by companies vary widely. These variances can distort comparisons between companies, industries, and countries, and make benchmarking difficult. Of course, within a single company this is not an issue.

Function points are now the most commonly used metric in both the United States and Europe for software projects, and are rapidly growing in usage throughout the world. Development productivity measurements normally use function points in two fashions: function points per staff month and/or work hours per function point [10, 11, 12]. For additional information on functional metrics, refer to the Web site of the non-profit International Function Point Users Group at <www.ifpug.org>.

The federal government, some military projects, and the defense industry still perform measurements using the older *lines-of-code* metric. This metric is hazardous because it cannot be used for measuring many important activities such as requirements, design, documentation, project management, quality assurance, and the like. There are also programming languages such as Visual Basic that have no effective rules for counting lines of code. About one third of the large software projects examined utilized several programming languages concurrently, and one large application included 12 different programming languages.

Measures of quality are powerful indicators of top-ranked software producers and are almost universal on successful projects. Projects that are likely to fail, or have failed, almost never measure quality. Quality measures include defect volumes by origin (i.e., requirements, design, code,

bad fixes) and severity level, defect severity levels, and defect repair rates.

Really sophisticated companies and projects also measure defect removal efficiency. This requires accumulating all defects found during development and also after release to customers for a predetermined time period. For example, if a project finds 900 defects during development and the users find 100 defects in the first three months of use, then it can be stated that the project achieved a 90 percent defect removal efficiency level. Of course, any defect found after the first three months lowers the defect removal value.

It is interesting that successful projects are almost always better than 95 percent in defect removal efficiency, which is about 10 percent better than the U.S. average of 85 percent [13].

It is not possible to measure defect removal efficiency for cancelled projects since there is no customer usage. However, for projects that finally get

**“... it can be
hypothesized that lack
of effective quality
control on large systems
is a major contributor
to both cost and
schedule overruns.”**

released to customers – although delivered late – defect removal efficiency seldom tops 80 percent, or about 5 percent below U.S. averages and 15 percent below successful projects. This statement is based on only about a dozen large systems because almost universally, projects that are delayed or over budget do not have effective quality measurements in place.

Since the bulk of schedule delays and cost overruns tends to occur during testing and is caused by excessive defect volumes, it can be hypothesized that lack of effective quality control on large systems is a major contributor to both cost and schedule overruns.

Successful and Unsuccessful Milestone Tracking

The phrase *milestone* tracking is ambiguous in the software world. It sometimes refers to the start of an activity, sometimes to

the completion of an activity, and sometimes to nothing more than a calendar date. In this article, the phrase refers to the point of formal completion of key deliverables or a key activity. Normally, a completion milestone is the direct result of some kind of review or inspection of the deliverable. A milestone is not an arbitrary calendar date.

Project management is responsible for establishing milestones, monitoring their completion, and reporting truthfully on whether the milestones were successfully completed or encountered problems. When serious problems are encountered, it is necessary to correct the problems before reporting that the milestone has been completed.

A typical set of project milestones for successful software applications in the nominal 10,000-function point size range would include completion of the following:

- Requirements review.
- Project plan review.
- Cost and quality estimate review.
- External design reviews.
- Database design reviews.
- Internal design reviews.
- Quality plan and test plan reviews.
- Documentation plan review.
- Deployment plan review.
- Training plan review.
- Code inspections.
- Each development test stage.
- Customer acceptance test.

Failing or delayed projects usually lack serious milestone tracking. Activities might be reported as finished while work was still ongoing. Milestones might be simple dates on a calendar rather than completion and review of actual deliverables. Some kinds of reviews may be so skimpy as to be ineffective.

Successful projects, on the other hand, regard milestone tracking as an important activity and try to do it well. There is no glossing over of missed milestones, or pretending that unfinished work is done. Delivering documents or code segments that are incomplete, contain errors, and cannot support downstream development work is not the way milestones occur on successful projects.

Another aspect of milestone tracking on successful projects is what happens when problems are reported or delays occur. The reaction is strong and immediate: corrective actions are planned, task forces assigned, and corrections occur as rapidly as possible. Among lagging projects, on the other hand, problem reports may be ignored and very seldom do corrective actions occur.

Successful and Unsuccessful Change Management

Applications in the nominal 10,000-function point size range run from 1 percent to 3 percent per month in new or changed requirements during the analysis and design phases [8]. This fact was discovered by measuring the initial function point totals at the requirements stage and comparing them to the function point total after design. If the initial function point total is 10,000 function points and the post-design total is 12,000 function points, then the overall growth is 20 percent. If the schedule for analysis and design took 10 calendar months, then the monthly growth rate was 2 percent per month.

The total accumulated volume of changing requirements can top 50 percent of the initial requirements when function point totals at the requirements phase are compared to those at deployment. Therefore, successful software projects in the nominal 10,000-function point size range must use state-of-the-art methods and tools to ensure that changes do not get out of control.

Successful change control for applications in the 10,000-function point size range include the following:

- A joint client/development change control board or designated domain experts.
- Using joint application design (JAD) to minimize downstream changes.
- Using formal prototypes to minimize downstream changes.
- Planned usage of iterative development to accommodate changes.
- Formal review of all change requests.
- Revised cost and schedule estimates for all changes greater than 10 function points.
- Prioritizing change requests in terms of business impact.
- Formal assignment of change requests to specific releases.
- Using automated change control tools with cross-reference capabilities.

One of the observed byproducts of using formal JAD sessions is a reduction in downstream requirements changes. Rather than having unplanned requirements surface at a rate of 1 percent to 3 percent per month, studies of JAD by IBM and other companies have indicated that unplanned requirements changes often drop below 1 percent per month due to the effectiveness of the JAD technique.

Prototypes are also helpful in reducing the rates of downstream requirements

changes. Normally key screens, inputs, and outputs are prototyped so users have some hands-on experience with what the completed application will look like.

However, changes will always occur for large systems. It is not possible to freeze the requirements of any real-world application, and it is naïve to think this can occur. Therefore, leading companies are ready and able to deal with changes, and do not let them become impediments to progress. Therefore, some form of iterative development is a logical necessity.

Successful and Unsuccessful Quality Control

Effective software quality control is the most important single factor that separates successful projects from delays and disasters. The reason for this is because finding and fixing bugs is the most expensive cost element for large systems and takes more time than any other activity.

Successful quality control involves both defect prevention and defect removal activities. The phrase *defect prevention* includes all activities that minimize the probability of creating an error or defect in the first place. Examples of defect prevention activities include JAD for gathering requirements, using formal design methods, using structured coding techniques, and using libraries of proven reusable material. The phrase *defect removal* includes all activities that can find errors or defects in any kind of deliverable. Examples of defect removal activities include requirements inspections, design inspections, document inspections, code inspections, and all kinds of testing.

Some activities benefit both defect prevention and defect removal simultaneously. For example, participation in design and code inspections is very effective in terms of defect removal, and also benefits defect prevention. The reason why defect prevention is aided is because inspection participants learn to avoid the kinds of errors that inspections detect. Successful quality control activities for 10,000-function point projects include the following:

Defect Prevention

- JAD for gathering requirements.
- Formal design methods.
- Structured coding methods.
- Formal test plans.
- Formal test case construction.

Defect Removal

- Requirements inspections.
- Design inspections.

- Document inspections.
- Code inspections.
- Test-plan and test-case inspections.
- Defect repair inspections.
- Software quality assurance reviews.
- Unit testing.
- Component testing.
- New function testing.
- Regression testing.
- Performance testing.
- System testing.
- Acceptance testing.

The combination of defect prevention and defect removal activities leads to some very significant differences in the overall numbers of software defects compared between successful and unsuccessful projects. For projects in the 10,000-function point range, the successful ones accumulate development totals of around 4.0 defects per function point and remove about 95 percent of them before customer delivery. In other words, the number of delivered defects is about 0.2 defects per function point or 2,000 total latent defects. Of these, about 10 percent or 200 would be fairly serious defects. The rest would be minor or cosmetic defects.

By contrast, the unsuccessful projects accumulate development totals of around 7.0 defects per function point and remove only about 80 percent of them before delivery. The number of delivered defects is about 1.4 defects per function point or 14,000 total latent defects. Of these about 20 percent or 2,800 would be fairly serious defects. This large number of latent defects after delivery is very troubling for users.

One of the reasons why successful projects have such high defect removal efficiency compared to unsuccessful projects is the usage of design and code inspections [14, 15]. Formal design and code inspections average about 65 percent efficiency in finding defects. They also improve testing efficiency by providing better source materials for constructing test cases.

Unsuccessful projects typically omit design and code inspections and depend purely on testing. The omission of upfront inspections causes three serious problems: (1) the large number of defects still present when testing slows the project to a standstill, (2) the *bad fix* injection rate for projects without inspections is alarmingly high, and (3) the overall defect removal efficiency associated with only testing is not sufficient to achieve defect removal rates higher than about 80 percent.

(Note: The term *bad fixes* refers to secondary defects accidentally injected by means of a patch or defect repair that is

itself flawed. The industry average is about 7 percent, but for unsuccessful projects the number of bad fixes can approach 20 percent; i.e., one out of every five defect repairs introduced fresh defects [13]. Successful projects, on the other hand, can have bad-fix injection rates of only 2 percent or less.)

Conclusions

There are many ways to make large software systems fail. There are only a few ways of making them succeed. It is interesting that project management is the factor that tends to push projects along either the path to success or the path to failure.

Large software projects that are inept in quality control and skimpy in project management tasks are usually doomed to either outright failure or massive overruns.

Among the most important software development practices leading to success are those of planning and estimating before the project starts, absorbing changing requirements during the project, and successfully minimizing bugs or defects.

Successful projects always excel in these critical activities: planning, estimating, change control, and quality control. By contrast, projects that run late or fail typically had flawed or optimistic plans, had estimates that did not anticipate changes or handle change well, and failed to control quality. ♦

References

1. Jones, Capers. Patterns of Software Systems Failure and Success. Boston, MA: International Thompson Computer Press, 1995.
2. Jones, Capers. Software Assessments, Benchmarks, and Best Practices. Boston, MA: Addison Wesley Longman, 2000.
3. Ewusi-Mensah, Kweku. Software Development Failures. Cambridge, MA: MIT Press, 2003.
4. Glass, R.L. Software Runaways: Lessons Learned From Massive Software Project Failures. Englewood Cliffs, NJ: Prentice Hall, 1998.
5. The Standish Group. The CHAOS Chronicles Vers. 3.0. West Yarmouth, MA: The Standish Group, 2004.
6. Yourdon, Ed. Death March – The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects. Upper Saddle River, NJ: Prentice Hall PTR, 1997.
7. Jones, Capers. Estimating Software Costs. New York: McGraw Hill, 1998.
8. Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality. 2nd ed. New York: McGraw Hill, 1996.
9. Kan, Stephen H. Metrics and Models in Software Quality Engineering. 2nd ed. Boston, MA: Addison Wesley Longman, 2003.
10. Garmus, David, and David Herron. Function Point Analysis – Measurement Practices for Successful Software Projects. Boston, MA: Addison Wesley Longman, 2001.
11. International Function Point Users Group. IT Measurement – Practical Advice from the Experts. Boston, MA: Addison Wesley Longman, 2002.
12. Jones, Capers. "Sizing Up Software." Scientific American Magazine 279.6 (Dec. 1998).
13. Jones, Capers. Software Quality – Analysis and Guidelines for Success. Boston, MA: International Thomson Computer Press, 1997.
14. Radice, Ronald A. High Quality Low Cost Software Inspections. Andover, MA: Paradoxicon Publishing, 2002.
15. Wiegers, Karl E. Peer Reviews in Software – A Practical Guide. Boston, MA: Addison Wesley Longman, 2002.

About the Author



Capers Jones is founder and chief scientist of Software Productivity Research LLC. He is an international consultant on software management topics, a speaker, a seminar leader, and author. Jones was formerly at the ITT Programming Technology Center in Stratford, Conn., where he was assistant director of Programming Technology. Prior to joining ITT, he was at IBM for a 12-year period in both research and managerial positions. He received the IBM General Product Division's outstanding contribution award for his work in software quality and productivity improvement methods. Jones has published 12 books on software project management topics and more than 200 journal articles. He has given seminars on software project management in more than 20 countries to more than 150 major corporations, government agencies, and military services.

Phone: (401) 789-7662

Fax: (401) 782-2755

E-mail: cjones@spr.com

COMING EVENTS

November 3-5

*Association for Computing Machinery
SenSys '04*
Baltimore, MD
www.cse.ohio-state.edu/sensys04

November 7-10

*Amplifying Your Effectiveness
AYE 2004*
Phoenix, AZ
www.ayeconference.com

November 8-13

*13th Conference on Information and
Knowledge Management CIKM 2004*
Washington, D.C.
www.ir.iit.edu/cikm2004

November 14-18

SIGAda 2004
Atlanta, GA
www.acm.org/sigada/conf/sigada2004

November 15-18

*4th Annual Capability Maturity Model®
Integration Technology Conference*
Denver, CO
www.sei.cmu.edu/cmmi/events/cmmi-techconf.html

November 15-19

*Software Testing Analysis and Review
STARWEST '04*
Anaheim, CA
www.sqe.com/starwest/

November 17-19

2004 Federal CTO Summit
Washington, DC
www.federalctosummit.com

April 18-21, 2005

*2005 Systems and Software
Technology Conference*



Salt Lake City, UT
www.stc-online.org

Catastrophe Disentanglement: Getting Software Projects Back on Track[®]

E.M. Bennatan

Advanced Project Solutions, Inc.

If you are responsible for a late and over-budget software project, you are not alone — software project overruns are all too common. But if serious problems have existed for quite a while and the situation is getting worse, not better, you may have a project catastrophe on your hands. At this point, there is no established rescue process to follow. Dealing effectively with an out-of-control project is as much an emotional challenge as it is a managerial and technical one. This article describes a 10-step process to disentangle a software project catastrophe and get it back on track.

In Spencer Johnson's "Who Moved My Cheese?" [1], the *little people* keep coming back to where the cheese used to be even though it is not there anymore. It is a natural tendency to continue doing what we have always done even when, to an outside observer, it no longer makes sense. This behavior is quite common when projects get into trouble. We keep plodding away at the project hoping that the problems will go away and the *cheese* will miraculously reappear. In all too many cases, it does not.

Just as the smart thing to do when a ball of twine seems hopelessly entangled is to stop whatever we are doing with it (otherwise the tangle gets worse), so it often is with a disastrous project: The longer we keep at it, the worse it gets. At some point, we need to halt all activity and reassess what we are doing.

Disastrous software projects, or *catastrophes*, are projects that are completely out of control in one or more of the following aspects: schedule, budget, or quality. They are by no means rare — 44 percent of surveyed development organizations report that they have had software projects cancelled or abandoned due to significant overruns, and 15 percent say that it has happened to more than 10 percent of their projects (see Figure 1).

But, obviously, not every overrun or quality problem means a project is out of control, so at which point should we define a software project as a catastrophe? What are the criteria for taking the drastic step of halting all activities, and how do we go about reassessing the project? Most importantly, how do we get the project moving again? The answers to these questions are the essence of the concept of *catastrophe disentanglement*.

When Is a Project a Catastrophe?

Organizations and projects vary to such an extent that there can be no universal crite-

ria for branding a software project a catastrophe. The expectations from mission-critical, life support, or banking software are significantly different than from most consumer- or Internet-based software applications. But experience shows that in virtually all cases, projects are in deep trouble if serious problems have existed for quite a while and the situation is getting worse, not better. How is this reflected in terms of schedule, budget, and quality?

Schedule

Software projects rarely or never strictly follow their schedule; delays often grow and shrink like an accordion. It is a sad reality that software project delays are an excessively common occurrence (see Figure 2¹). But we are not looking at just any delay; the issue here is to identify those projects where the delay is growing uncontrollably.

To determine if the delay is out of control, divide the total development schedule into 12 phases, and look at each of the last three. Has the delay steadily grown in each phase? Is the total delay now greater than three phases (i.e., 25 percent of the total project schedule)?

On a one-year schedule, for example, look at the last three months and ask the following questions:

1. Was the delay significant two months ago?
2. Was the delay even greater one month ago?
3. This month, has the delay grown again?
4. Has the delay growth been steady (that is, not two small delays and one major delay caused by an identifiable event)?
5. Is the total delay now greater than three months?

If the answer to these questions is yes, it is probably a good idea to halt the project and reassess it.

Budget

A project is a budget catastrophe if its remaining projected cost far exceeds what

the development organization is willing to pay for it. In software projects, major budget overruns are often the result of schedule overruns or of attempts to reduce schedule overruns (e.g., by adding staff). The following are points to consider:

1. Does the project schedule appear to be a catastrophe? If so, project cost projections have little value at this time.
2. If the project schedule appears to be under control, then extrapolate budget overruns for the past three phases up to the end of the most current project schedule (assume that every future phase will continue to exceed the budget at a similar rate). Is this a cost your organization can bear?
3. Do you have current feedback from the project's customers and users? Do you have updated market research data? Is the original cost/value analysis for this project still valid?

Quality

A software project is a quality catastrophe if (a) the list of serious quality problems has been substantial for three periods and is not decreasing, or if (b) customers/users who have evaluated the software that is being developed are exceptionally critical of it.

The project problem list is a good indicator of how serious the problems are. The list is commonly divided into (a) critical, (b) serious, and (c) minor problems. The following are points to consider:

1. Is the critical problem list growing? Are problems being resolved? How fast are new problems being added?
2. The second level of serious quality problems can also indicate the gravity of the situation if the list is particularly long and not getting any shorter.
3. Another indicator to monitor is how well the quality problem lists are being maintained. Are problems being categorized correctly? Are problems being removed prematurely from the list? Are new problems being withheld from the list?

[®] 2004 E.M. Bennatan. Advanced Project Solutions, Inc.

Severe quality problems (those that are either critical or most serious) are often difficult, if not impossible, to see in the early stages of a project. In fact, many severe quality problems emerge only toward the end of a project (and sometimes only after its release). Even the *last-three-phases* technique can be ineffective during the first half of a project because too often problem lists have not yet been compiled or well maintained.

But project quality issues can be monitored from the outset if there is someone whose job it is to do so. This means assigning an independent software quality assurance (SQA) professional to every project team as soon as the project is launched. For small development teams, one SQA professional can be responsible for two or three projects, though large projects should have their own indigenous SQA team.

Customer and user feedback is the best source for evaluating project quality. Unfortunately, it is sometimes difficult to get feedback until a project is close to release. For large projects, it is often worth investing in prototypes and pre-releases, thus getting preliminary versions of the software into the users' hands for early evaluation and feedback. This investment is like an insurance policy: It reduces the risk of major product quality issues – but at a cost.

The Project Is a Catastrophe – Now What?

The following 10 steps describe the process for disentangling a failing software project and getting it back on track. Because these steps intrude on the responsibilities of the team members – most especially the project manager – the process should be confined to getting the project back on track and nothing more. Ultimately, the new project plan must gain the unreserved support of the development team members, and the details should be left up to them.

1. Stop

Once you have determined that a software project is unlikely to be completed with any reasonable degree of success, the next step is painful but clear: Stop all activities immediately. This is a difficult decision because it will always be open to harsh criticism from some circles. It is also a tough decision because, as we have seen, there is really no airtight algorithm for determining that a project is a catastrophe. Ultimately, the decision is a combination of data analysis and management experience.

Stopping a project should never leave a team idle. There is much to do in preparing the project for assessment, including the following:

- Collecting and updating project documentation and data.
- Preparing status reports for each team member and each team.
- Bringing the project software to the nearest point (backward, not forward) for demonstration. This means that except for minor exceptions, no new code should be written and no new features should be added or integrated (otherwise there is a risk that the demonstration will take too long to prepare).
- Assisting the project evaluator.

In addition, other activities should be prepared and held in reserve such as training and assistance to other projects.

2. Assign An Evaluator

Virtually all software projects in trouble have strong emotional and political hallmarks that often produce passionate advocates and opponents. Therefore, the importance of using an *external* project evaluator cannot be overstated. This will increase the likelihood of getting an unbi-

Ten Steps to Disentangle a Software Project Catastrophe

1. Stop.
2. Assign an external, unbiased evaluator.
3. Evaluate true project status (what has been achieved and what has not).
4. Evaluate team capabilities.
5. Define new minimal goals and requirements with senior (executive) management and customers.
6. Determine if minimum goals can be achieved (if not, then abandon, find alternative to the project).
7. Rebuild the project team.
8. Perform high-level risk analysis.
9. Develop reasonable estimates.
10. Install an early warning system.

ased and unemotional evaluation. Whom should you choose? Ideally, you should assign a reliable, pragmatic, experienced, and successful project evaluator who (a) understands the project technology, (b) has good social skills, and (c) can reprioritize other responsibilities to allow sufficient time for the evaluation.

For very large projects, use an evaluation team of two or more evaluators but

Figure 1: Percentage of Abandoned or Cancelled Software Projects Due to Significant Cost or Time Overruns in the Past Three Years [2]

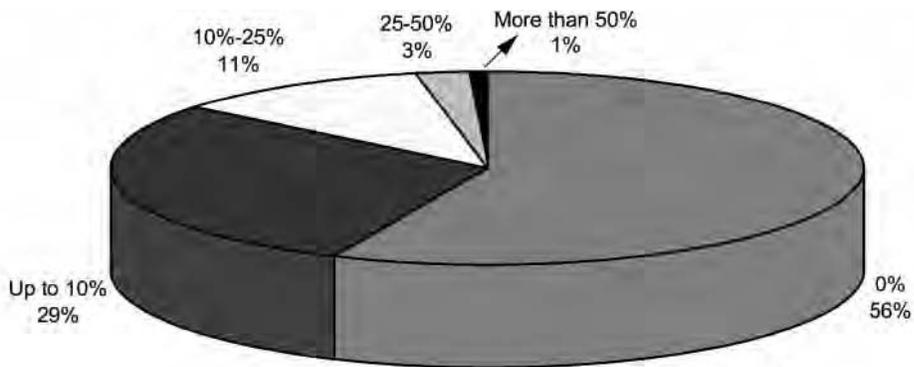
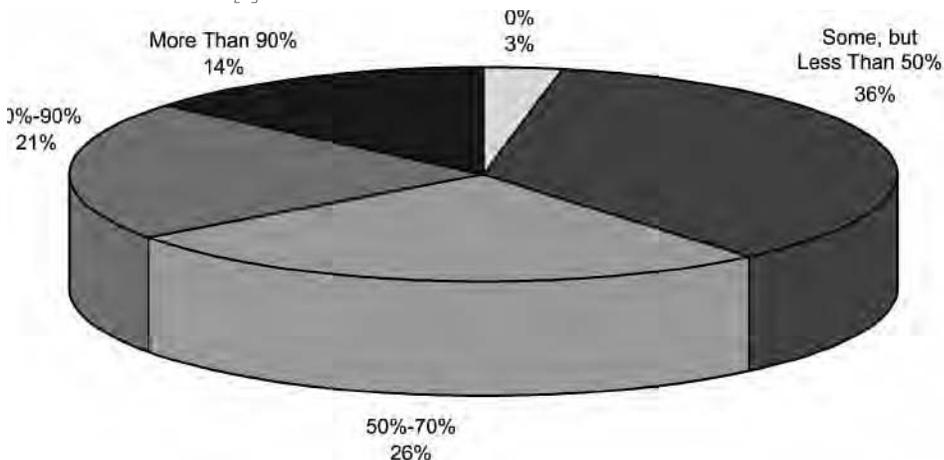


Figure 2: Percentage of Software Projects Completed Within 10 Percent of Original Estimated Time in the Past Three Years [2]



with a clearly designated chief evaluator.

3. Evaluate Project Status

The first challenge in evaluating a project is to determine its true status. Most failed software projects will have produced many status reports – some may even be quite positive – but they will not necessarily be objective or dispassionate. In establishing an unbiased view of the project status, do the following:

- Reduce tension by involving the project team in your evaluation and by being completely open (no secrecy or mysterious behind-closed-doors discussions).
- Consider only observable facts (e.g., not, “This feature used to work well but something has gone wrong.”).
- Consider accomplishments, not effort.
- For almost completed tasks, apply the 90-50 rule. (*It takes 50 percent of the time to do 90 percent of the work and another 50 percent to do the remaining 10 percent.*)
- Present your evaluation to the team before finalizing it and consider their responses (look for details and facts that you overlooked or misunderstood, while resisting undue pressure to amend your findings).

4. Evaluate the Team

Evaluating a team is a sensitive activity that should be handled both resolutely and tactfully. This step is purely part of the evaluation process and does not, at this point, result in any restructuring of the team. The following are questions to be considered:

- Does the project team have the necessary skill set and experience to successfully deliver the project?
- Do the team leaders have the leadership, technical skills, and the personality necessary to lead their team?
- Does the project manager have the required leadership, technical skills, and personality necessary to lead the project team, and does he or she command the respect of the team members?
- Are there any internal team conflicts or tensions that could disrupt the project?
- What is the level of team spirit and morale? If low, then why? (Are there reasons beyond the failing of the project?)

5. Define Minimum Goals

The emphasis here is on the word minimum; the project should be reduced to the smallest size that achieves only the most essential goals. This resetting of goals and objectives can only be performed with the active involvement of senior (executive) management and the customer². Divide all

project requirements into three sets:

- **Set One:** Essential requirements without which the project has no value.
- **Set Two:** Important requirements that greatly improve the project but are not essential.
- **Set Three:** Nice-to-have requirements that add to the project, but are not especially important.

Now, start by retaining the requirements from set one, and initially eliminating sets two, and three. This will often create tremendous opposition, but remember – we are dealing with a project that was totally out of control and may otherwise be cancelled. Occasionally, some elements from set two can be added, but this should be rare. All remaining requirements (from sets two and three) should be targeted for subsequent releases of the software.

Here is a word of caution: Be prepared to forestall the ploy by some stakeholders to second-guess the whole evaluation process by their insistence on listing all (or most) requirements in set one.

6. Can Minimum Goals Be Achieved?

The main challenge here is to determine whether the requirements in set one can reasonably be achieved. The questions to be addressed are the following:

- Is set one a genuine and significant reduction of the project scope?
- Is there a single requirement in set one that adds an order of magnitude to the complexity of the project? If so, are members of management aware of this and will they reconsider its inclusion³?
- Are the new project goals now achievable? Is there now a reasonable chance that the team will be able to deliver the requirements in an acceptable timeframe, within a reasonable budget, and with an acceptable quality level?
- How genuinely confident are the team members (and especially the project manager) in their ability to achieve the new set of goals?

If the minimum goals appear unachievable (and they are truly minimal), a recommendation to cancel the project may be the only remaining realistic course of action.

7. Rebuild the Team

Based on the evaluation of the team (see step four) it may be necessary to restructure and even partly re-staff the team to handle the new set of goals.

A halted software project can mean a team that is demotivated and demoralized. But in all probability, if the project was in deep trouble before it was halted, then the low morale did not start with the decision

to halt the project. However, the issue of team morale should be a major consideration in rebuilding the project team (this will be further discussed later).

In rebuilding the team, consider the following points:

- **Team Structure.** Is the project team structured optimally for the success of the project?
- **Team Functions.** Are the necessary team functions staffed?
- **Team Members.** Are there team members who should be replaced?
- **Team Leaders.** Are there team leaders who should be replaced?
- **Project Manager.** Is the project manager the right person to lead this project?

8. Risk Analysis

In all phases of a software development project, risk analysis is virtually an indispensable tool – this is particularly true of a failing project trying to get back on track. The process identifies risk events, mitigation steps and contingency plans, and assigns tracking responsibilities⁴.

High-level risk analysis (i.e., anticipating the most serious potential problems) should be performed as part of the project evaluation process. The analysis will not only help evaluate the chances of success in restarting the project, it will also help restore a level of confidence within the project team.

9. New Estimates and Schedule

Based on the minimal goals and the rebuilt team, new reasonable high-level estimates and a new schedule need to be prepared and the cost-effectiveness of the renewed project plan should be established. If the schedule is firm, ensure that budget, staffing level, and feature set are not also all fixed (or another catastrophe will ensue).

In many cases, it may be prudent to focus primarily on the schedule and feature set (the other parameters, such as budget and staffing levels, can initially be sidelined). This means that if the minimal feature set is firm, then calculate the project delivery date and vice versa. Remember that even a generous budget and an unrestricted staffing level may not be enough to resolve the problem of a fixed feature set with an uncompromising delivery date.

Here is a note on cost effectiveness: In analyzing the cost of completing a software project, only future costs (not costs already expended) should be considered. The cost of project completion should then be compared to the value of the completed project.

10. Establish Clear Project Review Milestones

Put in place an early warning system to ensure that the project does not slip back into catastrophe mode. Such a system should include the following:

- The introduction of an efficient and reliable project data collection and analysis system.
- Clear project evaluation criteria for management.
- A schedule of frequent project reviews with well-defined measurable milestones.

After successful completion of these project evaluation steps, and after determining that the renewed project plan is achievable and cost effective, the project can be restarted.

Case Study

A failing project is often like a hand in a cookie jar: to get some cookies out, you first have to let some go. Such was the case at Motorola with the software for a wireless telephony⁵ control and maintenance center (CMC) that we delivered several years ago as part of a 200,000-subscriber project to one of the emerging Eastern European countries (see [5]). The specially tailored CMC was a last minute add-on to the wireless telephony contract and was consequently not well defined.

The CMC was developed with a subcontractor team, based on an existing control system. The first phase of the project was devoted to producing a voluminous set of requirements, none of which could be omitted (according to the customer). The schedule was dictated – 16 months, which was set as close as possible to the date the subscriber telephony system was to become operational. Needless to say, every month was critical.

Five months into the project, key dates were already being missed. Seven months into the project, doubts began arising among senior management about whether the project would be ready on time. Nine months into the project, senior management was trying to calculate how much the late delivery penalties would cost, and a frantic marketing team was looking for alternatives. At all junctures, the development team was adamant that they would deliver the project on time.

At the end of nine months, amid significant resistance from the development and marketing teams, we brought the project almost to a complete halt (some tasks did continue). Two activities were then launched: (a) a total external review of the project, and, in parallel, (b) customer negotiations were reopened on the

CMC requirements.

- The project status was evaluated and it was confirmed that the then-current rate of progress would lead to a major project overrun. The team was moving forward at a steady pace but there was no way that they could meet the delivery date, or any date close to it.
- Because the CMC was critical for the operation of the whole system, the customer was cooperative in reevaluating the project's software features. Thus, a new set of minimal requirements was prepared.
- The project was rescheduled with two release dates: the first with the minimal feature set and the second with the remaining features.
- On the development team side, instead of using a single team for development, installation, and support, a cooperative effort was launched together with a local support team.
- Frequent project progress reviews were initiated by management with key development team members together with the customer.

As a result, a working CMC system was delivered on time and the full telephony system became operational as planned. The additional CMC features were provided as part of a later second release.

The Customer Perspective

Some software organizations' attitude toward customers is reminiscent of the librarian who disliked readers removing books from the library shelves because they disrupted the tidy placement of the books on the shelves. The librarian had confused means (the library) with goals (reading books). In software development, we also sometimes tend to confuse means (the project) with goals (customer satisfaction⁶).

There is justification for a project only as long as there are willing customers for its product. Hence, it is wise for both management and the project team to keep an ever-watchful eye on the customers: their needs, their expectations, and their opinion of the software being developed. After all, the continued development of a product that no longer has a willing customer (or user) is the ultimate project catastrophe.

Post-Project Reviews

Getting a failed software project back on track is an admirable accomplishment, but an even greater one is not having it go off track in the first place. Therefore, part of the catastrophe disentanglement procedure is preventing future recurrences of similar catastrophes. This is achieved

Warning Signs to Watch for in a Project:

- It is late and getting later.
- It is over budget and getting more so.
- Performance is poor and getting poorer.
- Criticism from customers/users is severe.

Choosing a Project Evaluator

- External (this might be the time to use a good consultant).
- Reliable, pragmatic, and experienced.
- Understands the project technology.
- Has good social skills.
- Can devote sufficient time.

The Post-Project Review

- What happened? (25%)
- Why did it happen? (25%)
- How to do better in the future? (50%)
- Who/what is to blame? (0%)

through a special review process held after the project has ended (successfully or otherwise).

The post-project review is a process intended to facilitate an understanding of why a project evolved the way it did. What was done right? What was done wrong? What can be done better next time? The review is a structured process that is not intended to find the guilty or to lay any blame, and is best done with a trained facilitator.

The output of the review includes a list of operational, procedural, and organizational changes and actions to ensure that mistakes are not repeated and successes are. In fact, the U.S. Army recommends that 50 percent of the review be devoted to discussions on how to do better in the future; the remaining time is devoted to what happened (25 percent), and why (25 percent) [7].

The Human Factor

The process of disentangling catastrophes is traumatic not just for the project team, but for the organization itself. Clearly, halting a project does not add to the motivation of a project team. Similarly, declaring a project to be a catastrophe does not add to the prestige of a development organization – though the courage to make such a decision often deserves praise.

While a highly motivated team is certainly one of the primary factors for project success, the fear of demotivating a team or tarnishing an organization's image

should never be a reason to allow a team to continue in the wrong direction. Catastrophe disentanglement should be viewed like corrective surgery: just as the body undergoes trauma in order to heal, so does the development organization.

One of the problems with the rather drastic measures of catastrophe disentanglement is that the knowledge that an organization will take such measures can inhibit the flow of accurate information (particularly bad news) to senior management. But successful corrective action, just like successful surgery, depends on the flow of truthful and accurate information even, in fact especially, when the news is bad.

The ability to bring bad tidings and make unpopular decisions is a desirable, if not entirely common, part of an organization's culture. Former Intel Chief Executive Officer Andy Grove said:

... If you are a middle manager you [may] face ... the fear that when you bring bad tidings you will be punished, the fear that management will not want to hear the bad news from the periphery. Fear that might keep you from voicing your real thoughts is poison. Almost nothing could be more detrimental to the well-being of the company. [8]

Grove's point is that effective corrective action requires accurate information – a reality not unfamiliar to those of us who drive a car: We cannot effectively steer a vehicle on the road if we cannot get accurate data. Thus, an organization that wants to be able to effectively evaluate its activities with processes such as the one described here, needs to promote the flow of accurate information by ensuring the following:

- The process is open and fair (not secretive).
- The staff is briefed about the process and the reason it is being adopted.
- The organization promotes a mistake-tolerant culture⁸. Blame and punishment need to be eliminated from the evaluation process (mistakes should be addressed in normal performance reviews alongside successes and achievements).

Conclusion

Most software catastrophes were troubled projects that went on for too long. Part of the trauma of dealing with them is the realization that “this shouldn't have happened,” or “we should have seen it coming.” Realizing this, the call to action is: “Something has to change around here.”

Returning to Johnson's “Who Moved My Cheese?” the tale continues:

The littlepeople were outraged, shocked, scared, and befuddled when the cheese disappeared. In their comfort, they didn't notice the cheese supply had been dwindling, nor that it had become old and smelly. They had become complacent. [1]

How better to describe the failing of a software project?◆

References

1. Johnson, Spencer, and Kenneth H. Blanchard. Who Moved My Cheese? An Amazing Way to Deal With Change in Your Work and in Your Life. Putnam Pub Group, 1998.
2. Bennatan, E.M. “The State of Software Estimation: Has the Dragon Been Slain?” Part 1. Executive Update 3.10. The Cutter Consortium, July 2002.
3. Brooks, Fredrick P. “The Mythical Man Month After 20 Years.” The 17th International Conference on Software Engineering, Seattle, WA, Apr. 23-30, 1995.
4. Bennatan, E.M. On Time Within Budget: Software Project Management Practices and Techniques. 3rd ed. John Wiley & Sons, 2000.
5. Bennatan, E.M. “Wireless Local Loop in Hungary – A Case Study.” New Telecom Quarterly 2nd Quarter, 1997 <www.tfi.com/pubs/ntq/auth-BennatanElli.html>.
6. Sullivan, Gordon R., and Michael V. Harper. Hope Is Not a Method. Times Business, Random House, 1996.
7. Meliza, Larry L. A Guide to Standardizing After Action Review (AAR) Aids. Report No. A348953. Orlando, FL: U.S. Army Research Institute, Field Unit, 1998 <www.stormingmedia.us/34/3489/A348953.html>.
8. Grove, Andrew S. Only the Paranoid Survive. HarperCollins Business, 1996.
9. Farson, Richard E., and Ralph Keyes. “The Failure-Tolerant Leader.” The Harvard Business Review 1 Aug. 2002.

Notes

1. To be statistically accurate, the results may have included some projects that were finished early, but we risked the speculation that such cases (if any) would only represent a small fraction of the results.
2. The term *customer* here refers to the entity that requested the project or that will

use its product, or more generally, for whom the project is being developed.

3. Fred Brooks [3] tells the story of a senior naval officer's last minute requirement after many months of negotiating features, schedule, and cost for a new navy helicopter. “It must be able to fly across the Atlantic,” he stated. Only after laboriously explaining to him the enormous complexity that it added to the project was the officer willing to drop the requirement.
4. For an overview of basic software project risk analysis, see [4].
5. *Telephony* here refers to the provision of telephone-related services.
6. Yes, profitability is usually a good goal, too.
7. A useful overview of a generic, after-action review process, which can be easily adapted for software projects, is given in [6].
8. For an interesting discussion of a mistake-tolerant business culture, see [9].

About the Author



E.M. Bennatan is president of Advanced Project Solutions, Inc., where he assists development companies in software project catastrophe disentanglement, introduction of orderly process into ad-hoc organizations, organizational structure, simplification of existing processes, and management of multinational development. Bennatan spent many years as senior director at Motorola leading multinational design centers and developing wireless access systems. He was also responsible for program management of Motorola's High Availability Systems corporate-wide initiative. Before Motorola, Bennatan spent several years developing defense and aerospace systems in the U.S. and overseas. Bennatan has authored several articles and books, including “On Time Within Budget: Software Project Management Practices and Techniques,” and is a senior member of the Institute of Electrical and Electronics Engineers and a member of the Association for Computing Machinery.

Advanced Project Solutions, Inc.
One Northfield Plaza
Northfield, IL 60093
Phone: (847) 441-3229
E-mail: bennatan@advancedps.com

Understanding Causal Systems

David N. Card
Software Productivity Consortium

This article describes a model and a supporting set of terms that facilitate reasoning about causal systems, planning for causal analysis, and designing process experiments. This perspective is based on practical experience in implementing causal analysis in industry. The implementation of effective causal analysis methods has become increasingly important as more software organizations transition to higher levels of process maturity where causal analysis is a required – as well as an appropriate – behavior.

A causal system is an interacting set of events and conditions that produces recognizable consequences. Causal analysis is the systematic investigation of a causal system in order to identify actions that influence a causal system, usually to minimize undesirable consequences. Causal analysis may sometimes be referred to as root cause analysis or defect prevention. Searching for the cause of a problem (*laying the blame*) is a common human behavior that would not seem to require much formalism. However, causal investigations often go wrong, beginning with the definition of a cause.

Causal analysis focuses on understanding cause-effect relationships. Three conditions must be established to demonstrate a causal relationship:

- First, there must be a correlation or association between the hypothesized cause and effect.
- Second, the cause must precede the effect in time.
- Third, the mechanism linking the cause to the effect must be identified.

The first condition implies that when the cause occurs, the effect is also likely to be observed. Often, this is demonstrated through statistical correlation and regression. While the second condition seems obvious, a common mistake in the practice of causal analysis is to hypothesize cause-effect relationships between factors that occur simultaneously. This is an over-interpretation of the correlational analysis.

Figure 1 shows a scatter diagram of two variables measuring inspection (or peer review) performance. These two variables frequently demonstrate significant correlations. This diagram and a correlation coefficient computed from the data often are taken as evidence that preparation causes detection.

However, most inspection defects are discovered *during* preparation. Both meters are running simultaneously. Thus, preparation performance cannot substantially influence detection performance. They are

measures of the same activity. Rather, the correlation suggests that some other factor affects both preparation and detection.

Issuing a mandate (as a corrective action) to spend more time in preparation may result in more time being charged to inspections, but it is not likely to increase the defect detection rate. The underlying cause of low preparation and detection rates may be a lack of understanding of how to prepare, schedule pressure, or other factors that affect both measures. That underlying cause must be addressed to increase both the preparation rate and detection rate. Recognition of the correlational relationship helps to narrow the set of potential causes to things that affect both preparation and detection performance.

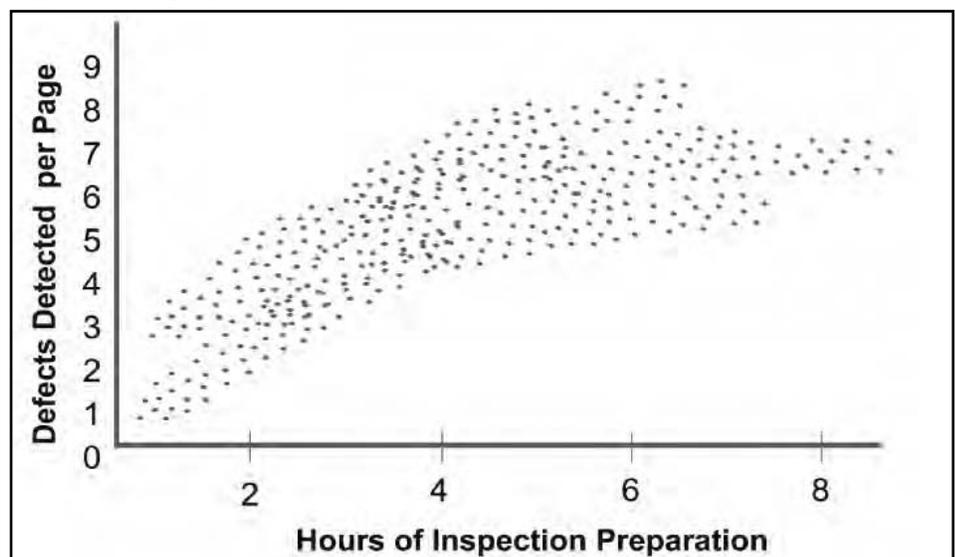
The relationship between the height and weight of adult human beings provides a good analogy to the situation described in Figure 1. Taller people tend to weigh more than shorter people. (Obviously other factors intervene as well.) While this is a necessary relationship, it is not a causal relationship. It would be a mistake to assume that increasing someone's weight would also increase

his/her height. Both variables are determined by other causes (chiefly genetics and childhood nutrition). Those underlying causes are the ones that need to be identified and manipulated in any causal system.

Some of the responsibility for this kind of misinterpretation can be attributed to statisticians. The horizontal and vertical axes of Figure 1 are typically referred to as the *independent* and *dependent* variables respectively. While these terms are simple labels, not intended to imply a causal relationship, they are often misunderstood.

Satisfying the third condition of a causal relationship requires investigating the causal system. Many good examples of causal analysis efforts in software engineering have been published [1, 2, 3, 4]. However, these efforts have adopted different terminology and approaches. In particular, the elements of a *causal system* have not been defined in a consistent way. The differences between the analysis procedures obscure the commonality in the subject matter to which the procedures are applied. Further complicating the situation are substantial differences in the

Figure 1: Example of Correlation Between Variables



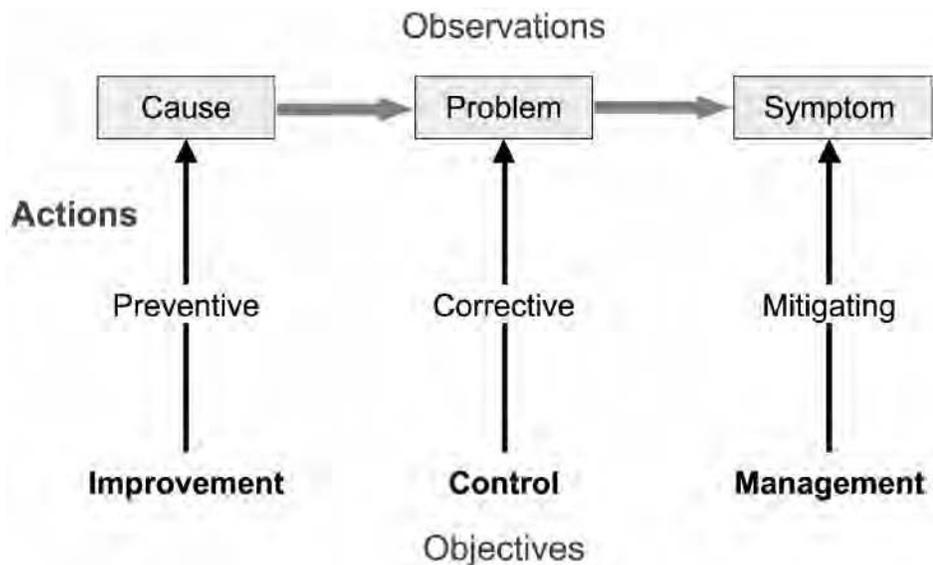


Figure 2: Elements of a Causal System

notion of causal analysis defined in the Capability Maturity Model® (CMM®) [5] and CMM IntegrationSM [6] (described later).

One of the consequences of a poor understanding of the nature of causal systems and causal analysis is that causal analysis sessions become superficial exercises that do not look deeply enough to find the important causes and potential actions that offer real leverage in changing performance. This reduces the cost benefit of the investment in causal analysis expected of mature software organizations. This article describes a model of causal analysis and a set of supporting terms that have evolved from extensive experience with the software industry. Some of these experiences with causal analysis were summarized in [7]. This experience encompasses scientific data-processing software, configuration man-

agement, and other software-related processes.

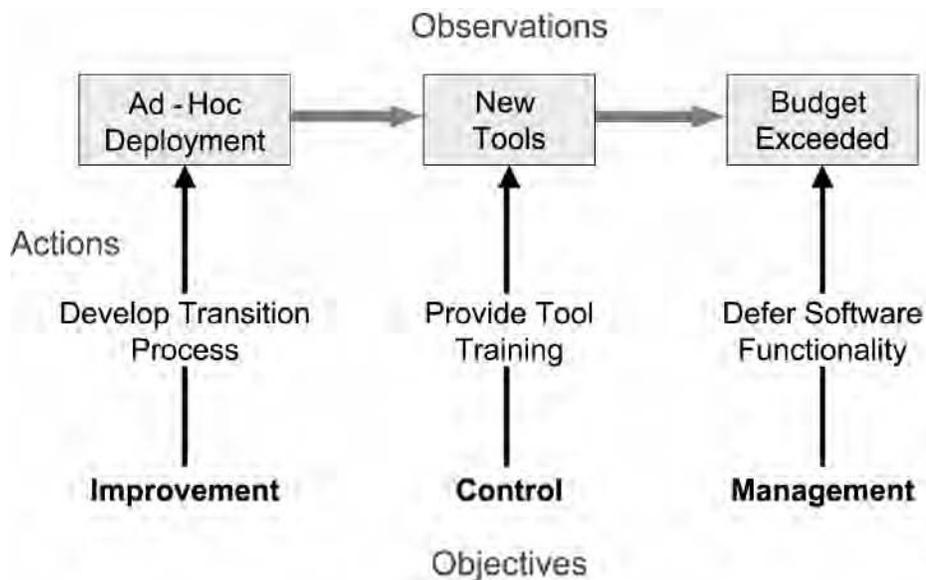
Elements of a Causal System

A cause-effect relationship may be one link in a potentially infinite network of *causes* and *effects*. A richer vocabulary than just causes and effects is needed to help us determine appropriate starting and stopping points for causal analysis. The model and terminology described in this section facilitate reasoning about causal systems and planning for causal analysis. Figure 2 describes the key elements of a causal system. Most of the approaches to causal analysis previously cited do not explicitly address all these elements of a causal system.

As indicated in the figure, causal systems include three classes of elements:

- **Objectives.** Our purposes in investigating the causal system.

Figure 3: Simple Example of a Causal System



- **Observations.** The events and conditions that comprise the causal system.
- **Actions.** Our efforts to influence the behavior of the causal system.

Observations are events and conditions that may be detected. Building an understanding of a causal system requires identifying these events and conditions, as well as discovering the relationships among them. Observations include the following:

- **Symptom.** These are undesirable consequences of the problem. Treating them does not make the problem go away, but may minimize the damage.
- **Problem.** This is the specific situation that, if corrected, results in the disappearance of further symptoms.
- **Cause.** These are the events and conditions that contribute to the occurrence of the problem. Addressing them helps prevent future similar problems.

Note that both problems and symptoms are effects of one or more underlying causes. Once a causal system is understood, action can be taken to change its behavior and/or impact on the organization. Actions may be of three types:

- **Preventive.** Reducing the chances that similar problems will occur again.
- **Corrective.** Fixing problems directly.
- **Mitigating.** Countering the adverse consequences (symptoms) of problems.

The corrective type usually includes actions to detect problems earlier so that they can be corrected before they produce symptoms. The optimum mix of preventive, corrective, and mitigating actions to be applied to a causal system depends on the cost of taking the actions as well as the magnitude of symptoms produced. Attacking the cause itself may not be the course of action that produces the maximum cost benefit in all situations. Potential symptoms and mitigations may be addressed as part of a risk-management activity.

Three objectives or motivations for undertaking causal analysis are common:

- **Improvement.** Triggered by recognition of an opportunity.
- **Control.** Triggered by an outlier or usual result relative to past performance.
- **Management.** Triggered by a departure from plans or targets.

Regardless of the motivation for causal analysis, all elements of the causal

® The Capability Maturity Model, CMM, and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
SM CMM Integration is a service mark of Carnegie Mellon University.

system (as described earlier) should be considered.

Most real causal systems are more complex than Figure 2 suggests. That is, a specific problem may produce multiple symptoms. Moreover, many causes may contribute to the problem. Consequently, many different actions of all types may be possible. The Ishikawa diagram [8] is a popular tool for describing and reasoning about causal systems.

These general concepts of causal systems can be applied to the investigation of any undesirable situation, not just to the investigation of defects. Figure 3 shows an example of a causal system explaining a cost problem.

In the hypothetical example of Figure 3, a project has exceeded its budget for the work accomplished to date. This is a *symptom* of an underlying problem. It might be overcome through management action by reducing the functionality of the software to be delivered, thus reducing the remaining work.

A causal analysis of the situation might reveal that the adoption of a new tool suite without preparation by the project had reduced productivity. That is the *problem*. Providing training might increase the efficiency and effectiveness of the team, returning productivity to its normal state.

Preventing future occurrences of such problems might be accomplished by establishing a formal process for deploying new technology that assures appropriate training is provided. Whether or not such an action is taken to prevent this cause, correct the problem, or mitigate the symptoms depends on their costs and expected benefits.

For example, if the project has already passed through the phase where the tool suite was expected to have the greatest impact, then providing training to this project team may not be cost-effective, although preventing future occurrences and mitigating the impact of the current problem may still be helpful.

CMM/CMMI Views of Causal Analysis

While the software community's interest in causal analysis predates the publication of the CMM [5], the pursuit of process maturity has become a primary motivation for the adoption of causal analysis practices today. Both the CMM and CMMI [6] contain process areas describing causal analysis activities. These are Defect Prevention (DP) and Causal Analysis and Resolution (CAR), respectively. These two views of causal analysis differ in three

important respects:

1. Required practices (activities).
2. Focus on prevention of defects.
3. Scope of triggering anomaly.

These differences are summarized below. The principal activities of the DP key process area of the CMM [5] are as follows:

- A DP plan is developed.
- Task kick-off meetings are held.
- Causal analysis meetings are held.
- Teams meet to coordinate actions.
- Defect prevention data is documented.
- Organizational process is revised.
- Project process is revised.
- Feedback is provided to staff.

Using the terminology described earlier, DP views *defects* as problems to be corrected. Failures are the consequences or symptoms of these problems that may have to be mitigated with workarounds, etc.

“One of the consequences of a poor understanding of the nature of causal systems and causal analysis is that causal analysis sessions become superficial exercises that do not look deeply enough to find the important causes and potential actions that offer real leverage in changing performance.”

The conditions that lead to the creation of defects are the causes to be prevented.

The specific practices of the CAR process area of CMMI [6] are as follows:

- Select defect data for analysis.
- Analyze causes.
- Implement the action proposal.
- Evaluate the effect of changes.
- Record data.

Note that DP requires some additional activities, not obvious in CAR. In particular, developing a DP plan and conducting task (usually phase) kick-off meetings exceed the explicit requirements of

CAR. DP is triggered by the recognition of an opportunity for improvement, e.g., a large number of defects associated with a particular activity.

DP focuses on developing preventive actions, rather than corrective or mitigating actions. Moreover, DP focuses on defects and their causes, not problems in general. Actions to detect defects earlier usually are considered *preventive* actions, although the case can be made that they really are corrective actions.

CAR is more general than DP. CAR *defines* a defect to include a broad range of problems. Any anomaly, outlier, or opportunity (as described in the preceding section) may trigger CAR, and result in any of the three types of actions identified earlier. It does not focus on prevention and early detection.

The generality of CAR makes it easy to come up with an example of investigating something to identify some kind of action, and thus claim satisfaction of the CAR requirements. On the other hand, systematic causal analysis does not need to be limited to the prevention of defects, as implied by DP. An understanding of the nature of causal systems helps to overcome the generality of CAR and ensure that each potential trigger for causal analysis is handled appropriately.

Summary

A good understanding of the basic concepts and terminology of causal systems helps to overcome the difficulties inherent in implementing a practice that seems *obvious*. The differences between the perspectives of CAR and DP has led to some problems as organizations either 1) try to facilitate the transition to CMMI by building a CMM Level 5 process that incorporates CMMI guidance into its initial design, or 2) transition an established CMM Level 5 organization to CMMI.

A causal analysis process based on CAR usually does not satisfy the CMM requirements for DP. A causal analysis process based on DP usually does not satisfy CMMI requirements for CAR. Understanding and applying the basic concepts of causal analysis underlying both process areas makes it possible to design a process that satisfies both sets of requirements.

Effective causal analysis is becoming even more important to the software industry as process maturity increases and new forces, such as Six Sigma [9] focus increasing attention on quality improvement. Academic researchers, especially those conducting empirical studies, also may benefit from thinking a little more



Get Your Free Subscription

Fill out and send us this form.

OO-ALC/MASE

6022 FIR AVE

BLDG 1238

HILL AFB, UT 84056-5820

FAX: (801) 777-8069 DSN: 777-8069

PHONE: (801) 775-5555 DSN: 775-5555

Or request online at www.stsc.hill.af.mil

NAME: _____

RANK/GRADE: _____

POSITION/TITLE: _____

ORGANIZATION: _____

ADDRESS: _____

BASE/CITY: _____

STATE: _____ ZIP: _____

PHONE: (____) _____

FAX: (____) _____

E-MAIL: _____

CHECK BOX(ES) TO REQUEST BACK ISSUES:

JULY2003 TOP 5 PROJECTS

AUG2003 NETWORK-CENTRIC ARCHT.

SEPT2003 DEFECT MANAGEMENT

OCT2003 INFORMATION SHARING

NOV2003 DEV. OF REAL-TIME SW

DEC2003 MANAGEMENT BASICS

MAR2004 SW PROCESS IMPROVEMENT

APR2004 ACQUISITION

MAY2004 TECH.: PROTECTING AMER.

JUN2004 ASSESSMENT AND CERT.

JULY2004 TOP 5 PROJECTS

AUG2004 SYSTEMS APPROACH

SEPT2004 SOFTWARE EDGE

TO REQUEST BACK ISSUES ON TOPICS NOT LISTED ABOVE, PLEASE CONTACT KAREN RASMUSSEN AT <STSC.CUSTOMERSERVICE@HILL.AF.MIL>.

systematically about causal systems. Application of the concepts and terminology presented here helps ensure that causal systems get fully investigated and effective actions are taken. ♦

References

1. Mays, R., et al. "Experiences With Defect Prevention." *IBM Systems Journal* Jan. 1990.
2. Dangerfield, O., et al. "Defect Causal Analysis – A Report From the Field." ASQC International Conference on Software Quality, Oct. 1992.
3. Yu, W. "A Software Fault Prevention Approach in Coding and Root Cause Analysis." *Bell Labs Technical Journal* Apr. 1998.
4. Leszak, M., et al. "Classification and Evaluation of Defects in a Project Perspective." *Journal of Systems and Software* Apr. 2002.
5. Paulk, Mark, et al. *Capability Maturity Model*. Addison-Wesley, 1994.
6. CMMI Development Team. *Capability Maturity Model – Integration, Vers. 1.1*. Pittsburgh, PA: Software Engineering Institute, 2001.
7. Card, D. "Learning From Our Mistakes With Defect Causal Analysis." *IEEE Software* Jan. 1998.
8. Ishikawa, K. *Guide to Quality Control*. Asian Productivity Organization Press, 1986.
9. Harry, M., and R. Schroeder. *Six Sigma*. Doubleday, 2000.

About the Author



David N. Card is a fellow of the Software Productivity Consortium, where he provides technical leadership in software measurement and process improvement. During 15 years at Computer Sciences Corporation, he spent six years as the director of Software Process and Measurement, one year as a resident affiliate at the Software Engineering Institute, and seven years as a member or manager of the research team supporting the NASA Software Engineering Laboratory. Card is the editor-in-chief of the *Journal of Systems and Software*. He is the author of "Measuring Software Design Quality," co-author of "Practical Software Measurement," and co-editor of "ISO/IEC standard 15939:2002 Software Measurement Process." Card is a senior member of the American Society for Quality.

Software Productivity Consortium
2214 Rock Hill RD
Herndon, VA 20170
Phone: (703) 742-7199
Fax: (703) 742-7200
E-mail: card@software.org

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:



Personal Software Process/ Team Software Process

March 2005

Submission Deadline: October 18, 2004

Personal Computing

April 2005

Submission Deadline: November 15, 2004

Configuration Management

June 2005

Submission Deadline: January 17, 2005

Please follow the Author Guidelines for CrossTalk, available on the Internet at <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on all software-related topics at any time, along with Letters to the Editor and BackTalk.



Requirements Engineering So Things Don't Get Ugly

Deb Jacobs
Focal Point Associates

Seasoned IT professionals remember those panicked moments when customers say, "That's not what we're looking for;" customer staff couldn't agree; requirements constantly changed; embarrassment when you couldn't develop requirements as promised; requirements are overlooked; estimates are skewed due to lack of understanding; and all the nice to have's drove cost and schedule. Requirements engineering is a tough task for both the requirements receiver and the requirements giver, even if you know exactly what you want. How you see a requirement depends on what vantage point you're coming from. We must understand both points of view – giver and receiver – to truly be able to do effective requirements engineering.

It was a cold, gloomy night in October. Outside the fog was so thick you could not see your hand in front of your face. Inside was even worse. Joe walked out of the conference room in a daze. He tried to remember what had happened but it all seemed like such a blur. The throbbing in his head grew even stronger. He hurried down the hall to the men's room. His thoughts were spinning. He asked himself, "What am I going to do?"

Joe had just learned the project he was working on would require more overtime. He kept thinking of how his wife had threatened to leave him just last week if he could not spend more time with her and the kids; in fact, he had not seen his kids in weeks. By the time he got home, they were in bed. Even when he did see them, he was so tired and frustrated he did not enjoy them. In fact, he had not really enjoyed life in a long time. It had actually started about a year ago when he had begun working on this project.

All of a sudden, the meeting came back to him. Voices screamed out in his head: "What do you mean that's not what you want? That's what the requirements say."

"That's not what we meant though. Don't you people understand anything?"

"We understand what you wrote down in the statement of work."

"But did you even bother to ask what we meant?"

"Well, we thought it was pretty clear."

"You missed the basic functionality we were looking for; in fact, this is so bad we're going to have to start completely over. And, by the way, we can't give you any slack on the schedule either."

For Joe, things were definitely getting ugly! This scenario may sound familiar to many of you. It happens time after time on project after project. So, how does it get like this?

How Does a Project Get Like This?

Sadly, for the information technology (IT) industry as well as their customers, studies show that the majority of systems are delivered with only about 42 percent to 67 percent of requirements. The Standish Group has found that even though projects are being delivered on time and within budget, the statistics for delivering requirements and meeting customer expectations are decreasing significantly [1].

Figure 1 shows a summary of The Standish Group's reports concerning project success as well as the top 10 most important elements for successful projects. The Standish Group stated,

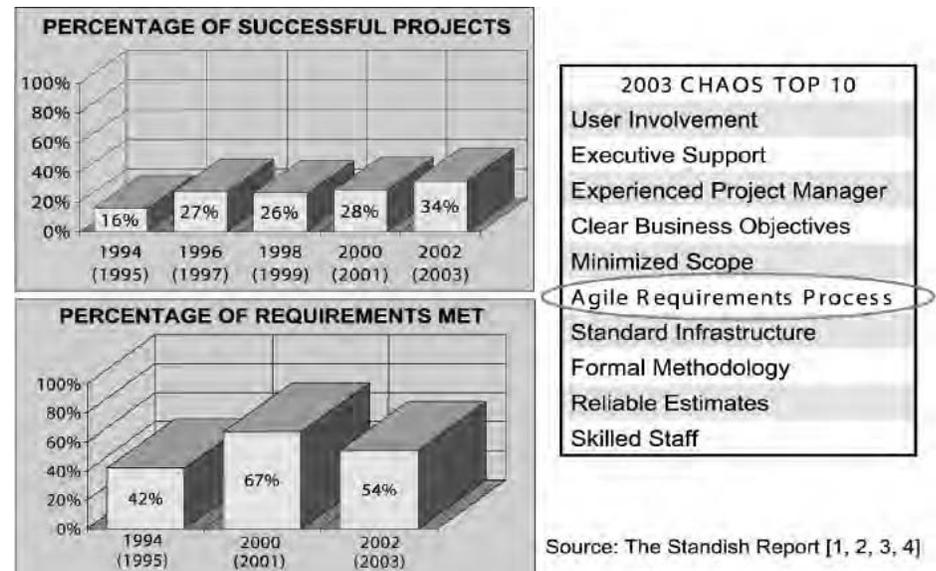
We find that on average only 54 percent, down from 67 percent in 2001, of the originally defined features of a project are delivered. Even more troubling is the realization that of those features that are

delivered – a full 45 percent are NEVER used. [4]

This article does not contain any new or eye-opening information; much of the information discussed is well known to requirements engineering experts. Best practices in requirements engineering have been honed since about 1968, and people have been writing about and teaching requirements engineering for several years; however, statistics continue to show that many IT practitioners and project managers still are not listening or have not been exposed to good requirements engineering.

In my positions over the last 20-plus years as project manager, software/technical project manager, software developer, systems engineer, process improvement engineer, new business proposal manager, and IT instructor, I have had the opportunity to be both the requirements giver and the requirements receiver. I have seen some very good examples of require-

Figure 1: Standish Group Findings Summary



If Architects Had to Work Like Programmers

Dear Mr. Architect:

Please design and build me a house. I am not quite sure what I need, so you should use your discretion.

My house should have between two and 45 bedrooms. Just make sure the plans are such that the bedrooms can be easily added or deleted. When you bring the blueprints to me, I will make the final decision of what I want. Also, bring me the cost breakdown for each configuration so that I can arbitrarily pick one.

Keep in mind that the house I ultimately choose must cost less than the one I am currently living in. Make sure, however, that you correct all the deficiencies that exist in my current house (the floor of my kitchen vibrates when I walk across it, and the walls do not have nearly enough insulation in them).

As you design, also keep in mind that I want to keep yearly maintenance costs as low as possible. This should mean incorporating extra-cost features like aluminum, vinyl, or composite siding. (If you choose not to specify aluminum, be prepared to explain your decision in detail.) Please take care that modern design practices and the latest materials are used in constructing the house, as I want it to be a showplace for the most up-to-date ideas and methods. Be alerted, however, that the kitchen should be designed to accommodate, among other things, my 1952 Gibson refrigerator.

To ensure that you are building the correct house for our entire family, make certain that you contact each of our children, and also our in-laws. My mother-in-law will have very strong feelings about how the house should be designed, since she visits us at least once a year. Make sure that you weigh all of these options carefully and come to the right decision. I, however, retain the right to overrule any choices that you make.

Please do not bother me with small details right now. Your job is to develop the overall plans for the house: Get the big picture. At this time, for example, it is not appropriate to be choosing the color of the carpet. However, keep in mind that my wife likes blue.

Also, do not worry at this time about acquiring the resources to build the house itself. Your first priority is to develop detailed plans and specifications. Once I approve these plans, however, I would expect the house to be under roof within 48 hours.

While you are designing this house specifically for me, keep in mind that sooner or later I will have to sell it to someone else. It therefore should have appeal to a wide variety of potential buyers. Please make sure before you finalize the plans that there is a consensus of the population in my area that they like the features this house has.

I advise you to run up and look at my neighbor's house he constructed last year. We like it a great deal. It has many features that we would also like in our new home, particularly the 75-foot swimming pool. With careful engineering, I believe that you can design this into our new house without impacting the final cost.

Please prepare a complete set of blueprints. It is not necessary at this time to do the real design since it will be used only for construction bids. Be advised, however, that you will be held accountable for any increase of construction costs as a result of later design changes.

To be able to use the latest techniques and materials and to be given such freedom in your designs is something that cannot happen too often. Contact me as soon as possible with your complete ideas and plans.

Respectfully,

J.P. Anonymous

P.S.

My wife has just told me that she disagrees with many of the instructions I have given you in this letter. It is your responsibility as the architect to resolve these differences. I have tried in the past and have been unable to accomplish this. If you cannot handle this responsibility, I will have to find another architect.

P.P.S.

Perhaps what I need is not a house at all, but a travel trailer. Please advise me as soon as possible if this is the case.

ments engineering, and I have seen very tragic requirements engineering. I have become passionate about this topic based on these good and bad requirements experiences.

While working off and on as a process improvement engineer and process manager over the last several years with various organizations, I have had the distinct opportunity to learn about and use many exceptional tools. Some of the best tools I have found include the ever-popular, very effective Software Engineering Institute's Capability Maturity Model® (CMM®) Integration (CMMI®) [5] and the ISO [International Organization for Standardization] 9001.

I have developed processes for accomplishing requirements engineering based on numerous resources throughout my career. These processes have been developed based on the IT industry best practices documented in the CMM and CMMI as well as my own personal lessons learned. The techniques and process discussed in this article are a culmination of these process development efforts. Each organization must tailor a process to fit its particular needs but this process will provide an idea of the various aspects of requirements engineering that should be addressed in a successful requirements engineering process.

The one thing I have learned well, and heard many times from other IT professionals, is that requirements engineering is tough work! For the requirements giver, it is very hard to articulate requirements either in writing or verbally, even if you know exactly what you want. It is just as difficult for the requirements receiver to understand what others are trying to articulate. We tend to overlook seeing things from others' points of view. When engineers and clients start working together and understanding each other's points of view, we will truly be able to do effective requirements engineering.

The bottom line is this: Development teams must understand what they are building, or they cannot build it. This is only achievable through teamwork – developer and client teamwork.

Whose Responsibility Is Understanding Requirements?

To correct this prevalent problem, the IT industry as a group has to depart from the them-and-us attitude that permeates the industry: it must be just *us*. The finger pointing must stop, and we must start

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

working as a team, both the requirements receivers and requirements givers. The current trend toward agile/eXtreme programming (XP) [6] consists of several practices that lend themselves to accomplishing better requirements engineering. One significant aspect of agile/XP is working closely with the client throughout the development process, which is called Active Stakeholder Participation.

The following are some ideas that have worked well for others:

- Bill of rights or stakeholder contract.
- Approval process for all requirements.
- Win-win negotiations meetings that negotiate requirements based on technology, environment, time, effort, and budget constraints.
- Requirements team training; i.e., same training for all team members.

It is the development team's responsibility to learn to balance the stakeholder needs and expectations. The needs are the identified requirements and the expectations are the unidentified requirements. Sometimes the expectations drive the full understanding of the identified requirements. If you understand what the customer is looking for in terms of their expectations, you gain insight into what they have identified as the real requirements. It is the customer's responsibility to articulate their expectations so that the development team fully understands what they are looking for in the resulting product.

For both the development team and the customer, there must be a clear understanding of who are the decision makers or final authorities for requirements. This includes someone who can do the following:

- Add or approve a new requirement.
- Change an existing requirement.
- Accept changes to requirements.
- Direct the developer or their manager.
- Determine if a requirement has or has not been met.
- Accept requirements as met or not met.

A graphical depiction can help immensely in defining and keeping track of who's who. These should be approved by the appropriate managers and distributed to the entire team. If a project has a communications plan, this is a good place to include these diagrams.

Why Is Requirements Engineering So Important?

We, as an industry, cannot afford the consequences of not doing requirements engineering effectively. The cost of incorrect, misunderstood, and not agreed upon requirements affects all of us in terms of time, money, and lost opportunities. The

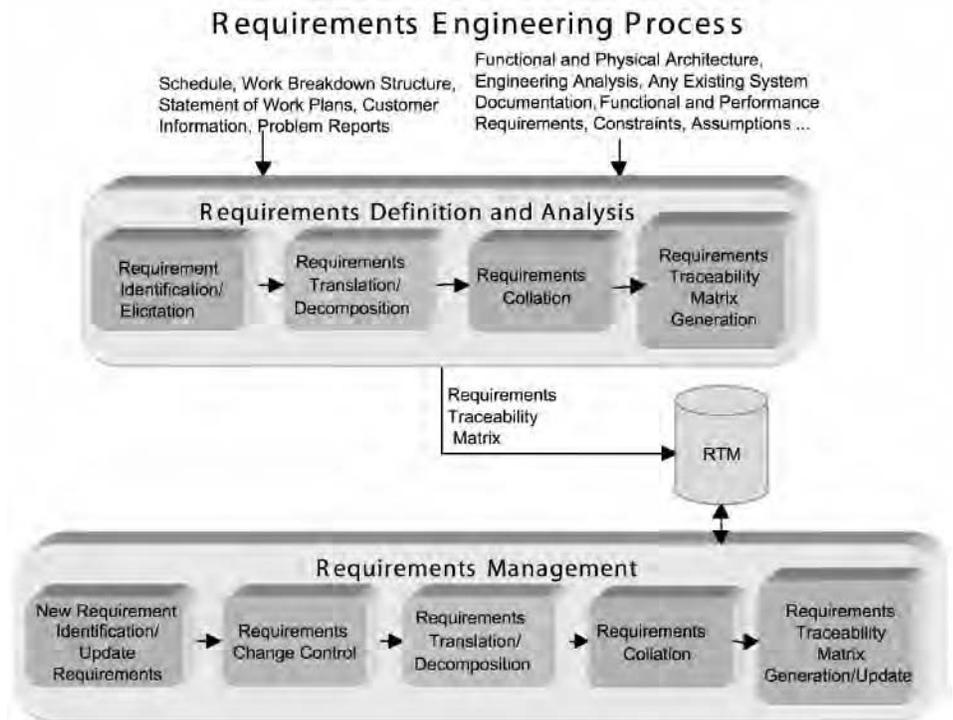


Figure 2: *Requirements Engineering Process*

results can be confusion, distrust, misdirection, frustration, lack of quality, higher cost, overtime, a general lack of understanding, and incapability due to being ill-equipped to handle issues.

Requirements engineering is a means of providing the functions and related characteristics of systems by providing the tools, concepts, and methods that mediate between the providers of information technology services and products, and the users or markets for the services and products. It is a means of providing the necessary communications to define needed products. Misunderstood, wrong, or even slightly skewed requirements propagate as the project moves forward until you get to the testing phase and scenarios like those discussed earlier occur.

Like dominoes, once problems start, they proliferate throughout the project – requirements problems at the beginning proliferate through design, development, and, finally, into test. Many times it gets to the point where starting over takes less time than trying to fix what you have already done. The sidebar “If Architects Had to Work Like Programmers” illustrates this point very well.

What Are Requirements?

Requirements tell the development team what the customer is contracting the team to build. As a whole, they provide a means of determining the functionality and attributes of the resulting product. The Institute of Electrical and Electronics

Engineers [7] defines a requirement as the following: (1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents; and (3) a documented representation of a condition or capability as in (1) or (2).

Proven Requirements Engineering Process

CMMI provides a good foundation for requirements engineering. It describes what should be included in an effective requirements engineering process. CMMI is based on best practices and lessons learned from the IT community, including both government-related and private industry. There are, in fact, several good taxonomies and methodologies that have been defined for requirements engineering. The requirements engineering process illustrated in Figure 2 and described in this article has proven effective on numerous successful projects and includes these basic best practices.

There are two major phases that are essential in defining and controlling requirements: Requirements Definition and Analysis, and Requirements Management.

Requirements Definition and Analysis

Requirements Definition and Analysis sets the stage for all subsequent tasks in devel-

oping the resulting product. Getting this right is key to the success of the overall project. A domino effect will begin here due to wrong, misunderstood, or slightly skewed requirements. The rule of thumb I have learned during my career is approximately 15 percent of project time should be spent on identifying, defining, and clarifying the requirements. This will vary depending upon the life-cycle methodology selected but it works well for most projects.

The following list includes some examples of typical inputs to the Requirements Definition and Analysis task. The inputs will depend upon an organization's processes, customer's processes, and whether the system being developed is an upgrade, major refurbishing of an existing system, or a new system.

- **Functional and performance requirements.** This information can be obtained using many methods but could simply be a list or a verbal exchange.

- **Statement of work.** Typically provided by the customer to the development team. Can be a key document for the customer and development team.
- **Plans.** Projects produce numerous plans that may drive some requirements such as the project plan, configuration management plan, logistics plan, communications plan, development plan, or engineering plans.
- **Customer information.** Various customer information can be derived that drives requirements such as customer standards, including user interface standards and security standards.
- **Problem reports.** Many times problem reports drive major system upgrades or refurbishment. The original problem report contains a good deal of information pertinent to understanding requirements.
- **Schedule.** The schedule may contain some information needed to understand what the customer is looking

for and the complexity expected, especially if the customer provides the overall schedule.

- **Work Breakdown Structure (WBS).** This provides information concerning the breakdown of requirements if it is developed using a WBS method that breaks the project down by product functionality.
- **Architecture (physical and functional).** For existing systems, this can be key to understanding requirements such as communications protocols, existing functionality, interfaces, etc. For new systems, the customer may provide this information in a statement of work and existing system documentation may help understand interfacing systems and data.
- **Engineering analysis.** Many times one or more trade studies or prototypes are developed that will help in understanding requirements.
- **Constraints.** There are many forms that constraints can take, including time, cost, and technical.
- **Assumptions.** Development team assumptions will drive the requirements and understanding of requirements. These should always be documented and discussed with the customer. Conversely, customers have assumptions that also must be communicated.
- **Existing system documentation.** Existing system documentation even when not up to date can provide invaluable information for understanding requirements.

Table 1: Requirements Identification Techniques

Technique	Brief Description
Interviews (structured and unstructured)	Meetings can be structured, unstructured, or a combination of both. <ul style="list-style-type: none"> • Structured: Interviewer has predetermined questions. • Unstructured: Interviewer may have some preliminary questions, but much less formal with random discussions.
Group Brainstorming	These are informal meetings that generate discussion of requirements. Basically used to generate ideas. Should have defined rules of conduct; avoid a free-for-all.
Observation	When upgrading systems, it is extremely helpful to observe a system in its operational environment. When the automated system is not available, observing the manual functions helps understanding.
Analysis of Existing Documentation	When upgrading systems, it is extremely helpful to analyze existing documentation (development and user documents). When the automated system is not available, some documentation may be available describing the functionality.
Questionnaires and/or Surveys	Questionnaires and surveys can give the development team insight into expectations (a word of caution: appropriate customer representative should always approve each requirement and derived requirement).
Prototyping	Prototyping is a great tool to use to simulate the final product especially for large systems by providing a visualization of the system or parts of a system. It is very effective in exposing any misunderstandings, risks, or missing functionality; clarifying confusing functionality; defining derived requirements; and avoiding being trapped in loops of refinement during design and coding. The two types of prototypes include discovery (aka throw-away) and production (aka evolutionary).
Conceptualization/Modeling	Conceptualization/modeling helps to clarify and prove requirements, weed requirements, clarify gaps in requirements, generate derived requirements, discover rules and procedures and functions/algorithms needed, ferret out data needs, and ensure correctness of the defined requirements. Effective in meetings (interviews and brainstorming) to describe how a system will behave, to describe interfaces (internal and external), and to discuss the consequences of each requirement using a whiteboard or paper.
Cognitive	Examining usability of requirements based upon cognitive characterizations of user interaction. This technique incorporates the human factors into requirements definition. Some methods include protocol analysis, laddering, card sorting, and repertory grids.

Requirements Identification/Elicitation

The Requirements Identification/Elicitation step provides an in-depth description of the desired resulting product. Some of the techniques used to identify and analyze requirements include those shown in Table 1.

I always recommend that one or more of these techniques be used to fully understand and communicate requirements throughout the project. The better the requirements are understood, the more likely the resulting system will be effective for the customer. During Requirements Identification/Elicitation, several questions need to be addressed that are shown in Table 2.

There is no simple formula for writing good, useable requirements; however, sources for writing good, useable requirements can be found on the Internet.

Requirements Translation/Decomposition

Once requirements have been identified, each requirement must be examined for a full understanding. Just as important is getting agreement between all project stakeholders, especially the identified decision makers. This step will be where implied or derived features/issues are uncovered. This is an iterative process where the interviewing and brainstorming sessions discussed in Table 1 are critical tools. These tools should continue to be used until all requirements are fully flushed out and beyond.

Drill down should be used to decompose requirements by starting at the basic high-level requirement and drilling down to the details of each requirement only after each level of detail is fully understood. Hence, only after the high-level requirements are fully understood and agreed upon should a development team move to finer details. Drill down should be iterative; as more details of higher-level requirements are understood, they are drilled down to lower levels for a complete understanding of what the customer is looking for. Drill down ensures that time and money are not wasted on detailing requirements that are misunderstood from the beginning.

During this step, requirements should be associated with a particular subsystem.

Requirements Collation

Grouping and prioritizing requirements are key to managing them. Requirements should be grouped for easier understanding, assignment, allocation, and tracking. The categories should be based upon the project needs; some suggestions include the following: function, effect of result, cause, impact and priority, timing, exception handling, and performance criteria. Function is the most prevalent and understandable categorization method.

Prioritization should be given to each requirement to understand its importance. Prioritization will help determine the sequence of tasking as well as weed out the essential versus the desirable versus the optional requirements.

Each requirement should be examined to determine any technical, cost, or schedule implications or risks. Any risks associated with each requirement should be recorded and tracked using the project's risk management process. The impact and the potential for occurrence will be key factors in managing risk.

Any impractical and excessive requirements should be weeded out since they

drive cost and schedule. Multiple requirements pertaining to the same functional/performance feature should be examined to ensure they are coherent and consistent. Finally, requirements should be allocated to system components for assignment. This can be done using a WBS if it has been designed using that method.

Requirements Traceability Matrix Generation

A very useful tool in managing requirements is a Requirements Traceability Matrix that is generated with the complete set of requirements. The matrix provides an authorized record of the requirements. The tool selected for the matrix will depend upon the size and scope of the project. A database is the optimal method for managing requirements but a simple spreadsheet can also be very effective. Several very good commercial tools are available; see <www.incose.org/tools/tooltax/reqtrace_tools.html> [8].

Creation of a homegrown requirements database will give users exactly what they are looking for if the expertise and time are available. Requirements tools have many advantages over a simple listing, including easy search, smooth requirements management, requirements change control, requirements metrics collection with minimal effort, and any needed documentation. The key to selecting the right tool is to ensure that you are getting the bang for the buck.

In [4], The Standish Group states, "Only 5 percent of new and changing applications will use a requirements management tool." That could be why we have many of our requirements problems.

Requirements Management

The requirements management phase consists of monitoring and controlling the requirements throughout the remaining development life cycle. Monitoring and controlling requirements ensures that the resulting system has all of the agreed upon or authorized requirements. It helps to avoid the widespread requirements epidemic known as requirements creep.

Requirements creep can drive both cost and schedule significantly. When a new or upgraded requirement is identified, the development team must go through the same process as defined for the Requirements Definition and Analysis phase with the appropriate decision makers.

New Requirement Identification/Update Requirements

As new requirements are identified or existing requirements change, they must be updated in the requirements baseline. Requirements should be maintained and baselined using the same type of configuration management controls as software such as the four elements of configuration management: identification, change management (the key), status accounting, and verification and audit.

Some configuration management tools have built-in requirements management features. This ensures the integrity of the

Table 2: *Requirements Engineering Process*

✓	What functions will the system perform? <ul style="list-style-type: none"> • Basics first. • Then drill down to details.
✓	What data will be input and output from resulting system? Are there any data format constraints? <ul style="list-style-type: none"> • External systems expecting data may need it in a specific format. • Data may be ported from an existing system for use in the upgraded/replaced system.
✓	Are there any system constraints? <ul style="list-style-type: none"> • Language. • Operating system. • Platform. • Tools/commercial off-the-shelf. • Web-based versus client/server versus mainframe versus ...
✓	What are the external interfaces? <ul style="list-style-type: none"> • Systems/subsystems. • Data (input or output).
✓	Are there any desired performance/reliability constraints? <ul style="list-style-type: none"> • Timing of throughput. • Limited down times. • Exception handling. • Better, faster, cheaper.
✓	Are there any customer standards that must be followed? <ul style="list-style-type: none"> • Quality standards. • Regulations. • Security (authentication, accessibility, physical, etc.). • Safety. • Format (look and feel).
✓	What is the level of technology desired? <ul style="list-style-type: none"> • Emerging technology usage versus older, more proven technologies. • May depend on customer's dollars available and risk sensitivity.
✓	What is the business objective for developing the work product? <ul style="list-style-type: none"> • Why is the organization interested in this development? What objectives does it support? • Priority of the work product for the customer. • Helps determine how much customer support will be available during development.

requirements. As requirements change – and they will – the changes must be controlled.

Requirements Change Control

Either formal or informal change control methods can be used. Formal change mechanisms include using a Configuration Change Board and appropriate formal authorizations. Less formal methods can also be effective, especially for smaller projects.

Whether a formal or informal change control method is selected, it is important that the identified decision makers finalize and authorize all requirements changes. This includes management of even the smallest detail since even a slight change can alter elements of the product. The changes must be coordinated with all stakeholders since they may have an impact on the tasks they are assigned either directly or indirectly.

Recycle Definition and Analysis

Once the new or upgraded requirements are approved, they must undergo the same process as the initial requirements.

- Requirements Translation/Decomposition.
- Requirements Collation.
- Requirements Traceability Matrix Generation/Update.

It is important to ensure that changes to requirements do not impact other requirements. Many times even simple changes will have a ripple effect on other requirements. The development team must be prepared to handle these changes. The Requirements Traceability Matrix should always reflect the current requirements as they are at any point in the project. They will be the authorized record for the resulting product.

Requirements Volatility Metric

Several metrics will help determine the status of a project but a key metric is requirements volatility, which is considered a key project success indicator. It indicates the stability of the baselined requirements.

How much change is too much and at what stage of the development cycle will it have a significant impact? There are some rules of thumb for how much requirements change is too much. “A Gentle Introduction to Software Engineering” indicates,

The accepted requirements volatility metric is 1 percent of requirements per month. If it is much less, one should ask oneself if the system

would be desirable to its intended audience. If it is much more than 2 percent a month, development chaos is all but assured. [9]

Other sources also use that rule of thumb; however, a study accomplished at the Colorado State University, Department of Computer Science concluded the following:

All the results show that changes have more influence on defect density when they occur closer to the end of the testing effort. This temporal dependence is generally exponential. Changes made very early can be relatively inconsequential, but those occurring later can raise defect density quite significantly. [10]

I have found in my experience that frequent changes to requirements are expected during the early stages of the project; however, a high volume of changes late in the development life cycle can have a significant impact to functionality, interfaces, cost, and schedule. The amount of acceptable requirements change can depend upon many factors, including the project phase, development team, requirements complexity, system complexity, system size, customer expectations, schedule, technology, methodologies, tools, etc.

If frequent changes are expected, it may be beneficial to use either an iterative build life cycle such as the spiral or incremental build or an agile/XP approach. There is an upside and a downside to all methods; the key is to select the method that is right for that development team, the customer, the system being developed, and the environment.

The Bottom Line

Time after time, projects experience nightmare scenarios similar to the one described at the beginning of this article. It is key to a project's success in delivering the customer's needed functionality that development teams and customers work as a team to develop effective requirements in order to develop effective products. If we look at things from each other's vantage point, the chance of success grows by leaps and bounds. We all look at things differently based on our background, education, experience, and simply from where we are standing at the moment. Open communications and respect for each other's position is crucial.

There is enough to panic about when developing a system without the added

stress of misunderstandings. Products must be delivered with better numbers than 42 percent to 67 percent of the required functionality. Nobody can afford the consequences of wrong, misunderstood, or even slightly skewed requirements.

The bottom line is this: Always remember that what you see is relative to where you are standing. We must all work as a team and select the best methodologies, techniques, and tools that keep things simple so things do not get ugly. ♦

References

1. The Standish Group. CHAOS: A Recipe for Success. West Yarmouth, MA: The Standish Group International, Inc., 1999 <www.standishgroup.com/sample_research/PDFpages/chaos1999.pdf>.
2. CHAOS Reports <www.standishgroup.com>.
3. The Standish Group. The CHAOS Report (1994). West Yarmouth, MA: The Standish Group International, Inc., 1995.
4. The Standish Group. What Are Your Requirements? West Yarmouth, MA: The Standish Group International, Inc., 2003, Standish Group (based on 2002 CHAOS Report).
5. CMMI Product Team. CMMISM for Systems Engineering/Software Engineering, Vers. 1.1, Staged Representation. Pittsburgh, PA: Software Engineering Institute, Dec. 2001.
6. The Official Agile Modeling (AM) Site <www.agilemodeling.com>.
7. Institute of Electrical and Electronic Engineers. IEEE Software Engineering Standards Collection: 1994 Edition. Washington, DC: IEEE 1994.
8. International Council on Systems Engineering <www.incose.org>.
9. Cook, David A., Leslie Dupaix, and Larry Smith. “A Gentle Introduction to Software Engineering.” Rev. 3.0. Hill Air Force Base, UT: Software Technology Support Center, 31 Mar. 1999.
10. Gotel, Orlena C.Z., and Anthony C.W. Finkelstein. “An Analysis of the Requirements Traceability Problem.” London, England: Imperial College of Science, Technology, and Medicine <http://csis.pace.edu/~ogotel/papers/RT_PAP.pdf>.

Additional Reading

1. Nuseibeh, Bashar, and Steve Easterbrook. “Requirements Engineering: A Road Map.” 3rd International Symposium on Requirements Engineering, Toronto, Canada, 2000

- <www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf>.
2. Requirements Working Group of the International Council on Systems Engineering. "Characteristics of Good Requirements." INCOSE Symposium.
 3. Bernard, Frederick R. Printers' Ink. Mar. 1927.
 4. Malaiya, Yashwant K., and Jason Denton. "Requirements Volatility and Defect Density." Ft. Collins, CO: Colorado State University.
 5. Bamford, Robert, and Bill Deibler. SSQC. Requirements Engineering Workshop <www.ssqc.com>.
 6. Ambler, Scott. The Elements of UML Style. Cambridge University Press, 18 Nov. 2002.
 7. Christel, M., and K. Kang. Issues in Requirements Elicitation. Pittsburgh, PA: Software Engineering Institute, 1992.
 8. McConnell, Steve. Construx Software. <www.stevemcconnell.com> or <www.construx.com>.
 9. Robertson, Suzanne, and James Robertson. Mastering the Requirements Process. Addison-Wesley, 1999.
 10. Hooks, Ivy. Writing Good Requirements. Proc. of the Third International Symposium of the NCOSE.

Vol. 2, 1993. Updated Sept 2003.

11. Wieggers, Karl. Writing Good Requirements. 2nd ed. Microsoft Press, 26 Feb. 2003.

12. KPMG. "What Went Wrong? Unsuccessful Information Technology Projects." KPMG Study <www.kpmg.ca>.

About the Author



Deb Jacobs is a professional consultant for Focal Point Associates specializing in process improvement and project management. She provides support to organizations in training, process improvement consulting, project management consulting, software engineering consulting, and proposal development. Jacobs has more than 25 years experience in system/software engineering, project management, process improvement, and proposal development. Her notable successes include leading a successful Capability Maturity Model® (CMM®) Level 3 effort in one year, successfully reorganizing struggling projects, mentoring new managers, and gaining new business for companies through proposal development.

She is former *SPINOUT* editor/originator; former Computer Emergency Response Team conference chairperson, infotec deputy Software Tracks chair, and a Software Engineering Institute CMM IntegrationSM contributor. She is currently working on a book to help organizations successfully achieve process maturity at minimal costs. Jacobs has a Bachelor of Science in computer science.

Focal Point Associates
c/o Priority Solutions
1508 JF Kennedy DR STE 100
Bellevue, NE 68005
Phone: (402) 932-5349
(402) 292-8660
E-mail: djacobsfpa@aol.com
djacobs@prioritytech.com
djacobs@sessolutions.com

WEB SITES

Project Management Institute

www.pmi.org

The Project Management Institute (PMI) is a not-for-profit, project-management professional association with more than 100,000 members in 125 countries. PMI publishes "A Guide to the Project Management Body of Knowledge," offers Project Management Professional certification, and maintains ISO 9001 certification in Quality Management Systems.

Software Program Managers Network

www.spmn.com

The Software Program Managers Network (SPMN) is sponsored by the deputy under secretary of defense for Science and Technology, Software Intensive Systems Directorate. It seeks out proven industry and government software best practices and conveys them to managers of large-scale Department of Defense software-intensive acquisition programs. The SPMN provides consulting, on-site program assessments, project risk assessments, software tools, guidebooks, and hands-on training.

NASA Independent Verification and Validation Facility

www.ivv.nasa.gov

The NASA Independent Verification and Validation (IV&V) Facility was established in 1993 to provide the highest achievable levels of safety and cost-effectiveness for mission critical software. The IV&V Facility's efforts have contributed to the

improved safety record of NASA since its inception. The IV&V Facility houses more than 150 full-time employees and more than 20 in-house partners and contractors.

Practical Software and Systems Measurement

www.psmc.com

Practical Software and Systems Measurement (PSM) is sponsored by the Department of Defense and the U.S. Army. PSM is an information-driven measurement process that addresses the unique technical and business goals of an organization by providing objective information needed to successfully meet cost, schedule, and technical objectives.

International Society of Parametric Analysts

www.ispa-cost.org

The International Society of Parametric Analysts (ISPA) is a professional society dedicated to the improvement and promotion of parametric cost modeling techniques and methodologies and the related fields of risk analysis, econometrics, design-to-cost, technology forecasting, and management. ISPA provides a forum that encourages the professional development of its members through the interchange of ideas and perspectives. ISPA members represent government agencies, universities, and nearly 200 organizations in 12 countries.



Independent Estimates at Completion – Another Method

Walt Lipke
Tinker Air Force Base

This article reviews the most frequently used Earned Value Management formulas for calculating the Independent Estimate at Completion (IEAC). The formulas are examined and discussed with reference to the findings from several studies. Some formulas appear to be inconsistent with the determinations from the studies of the Cost Performance Index (CPI). An alternative method of calculating IEAC is proposed, which is in agreement with the generalizations and conclusions from the IEAC and the CPI studies. This method shows promise.

The calculation of the Independent Estimate at Completion (IEAC) is significant to project management. It is a quick method facilitated by using Earned Value Management to predict the final project cost. Project managers (PM) and cost analysts often use IEAC to validate the bottoms-up forecast made by contract sources. When the IEAC result is substantially different from the contractor's estimate, more than likely the PM will question the discrepancy. The PMs also use the IEAC to justify continuation of the project to upper management. Thus, you can see IEAC has far reaching implications.

During the last 10 years, primarily due to the interest generated from the cancellation of the Navy's A-12 Avenger acquisition program, studies of the predictive accuracy of the various methods for calculating IEAC have been made. These studies considered and included several IEAC formulas and regression calculation methods. In general, no single specific method has been shown to be superior.

Although no particular method provided accurate results for all periods or phases of a project, some fundamental characteristics were observed. With the establishment of these characteristics, the application of some of the IEAC formulas and calculation methods appears to be questionable. However, these fundamentals have provided inspiration for proposing new IEAC methods in this article.

Studies of IEAC and Cost Performance Index

There are several popular formulas for calculating IEAC. In general, the equations use the cost to date added to the forecast cost for the work remaining. For the formulas identified here, the Cost Performance Index (CPI) and Schedule Performance Index (SPI) are the cumulative values unless otherwise noted¹. The following are the IEAC formulas most often seen and used:

- $IEAC_1 = ACWP + (BAC - BCWP) / CPI$
- $IEAC_2 = ACWP + (BAC - BCWP) / SPI$
- $IEAC_3 = ACWP + (BAC - BCWP) / (SPI * CPI)$
- $IEAC_4 = ACWP + (BAC - BCWP) / (wt_1 * SPI + wt_2 * CPI)$
- $IEAC_5 = ACWP + (BAC - BCWP) / CPI_x$

“Using the range of outcomes for the CPI from the statistical method and IEAC₁, a range for the estimates at completion can be calculated. The range may be computed for any statistical confidence level desired ...”

For IEAC₃ the product, SPI * CPI, is sometimes identified in literature as SCI. The abbreviations wt₁ and wt₂ of IEAC₄ are numbers between 0.0 and 1.0 used to weight the influence of the two indexes; the sum of wt₁ and wt₂ is equal to 1.0. The CPI_x in IEAC₅ is the cumulative value of the last x performance periods.

Two studies were performed in the 1990s that examined the prediction capability of the various formulas and regression methods [1, 2]. The generalizations and conclusions reached by these studies of IEAC are as follows:

1. The accuracy of regression-based forecasting has not been established. A

recommendation was made to further study the method.

2. The accuracy of index-based formulas depends upon the system in development, and the stage and phase of the project. The formula most frequently appearing in the tabulated results, regardless of type and stage, is IEAC₃.
3. The index-based formulas, including SPI are better applied early in the project. For projects behind schedule, SPI falsely improves as percent complete increases. Thus, the influence of SPI on the computation is not in agreement with actual schedule performance.
4. The accuracy of IEAC₄ with wt₁ = 0.2 and wt₂ = 0.8 is not supported.
5. The accuracy of IEAC₅ is better for middle and late stages of the project.

A second set of studies was performed that examined the behavior of the CPI throughout the life of a contract [3, 4, 5]. Two of the studies are very recent – spring and winter 2002 – and performed the analysis using statistical hypothesis testing. The three studies provided the following to PMs for assessing the validity of estimates at completion:

1. The result from IEAC₁ is a reasonable estimate of the lower bound of the final cost.
2. The cumulative value of the CPI stabilizes by the time the project is 20 percent complete. Stability is defined to mean that the final CPI does not vary by more than 0.10 from the value at 20 percent complete.
3. The value of the CPI tends only to worsen from the point of stability until project completion.

Commentary

The understanding of the behavior of the CPI, over the life of the project, provides insight regarding the study results for the IEAC equations. For IEAC₂,

IEAC₃ and IEAC₄, the divisor containing expressions of the present cumulative values of the SPI and CPI correlate to the final CPI, when the project is not performing as planned. We know from the study of CPI behavior that the final CPI is likely to be less than the present value; thus, having a SPI less than 1.0, or less than the CPI, will cause the estimate at completion to be larger than the result from using IEAC₁, as we know it must be. Regarding IEAC₅, it makes sense that this equation is a reasonably good predictor because the CPI_k is constructed from recent data. However, the limited amount of data used for creating the CPI_k causes IEAC₅ to oftentimes exhibit erratic behavior.

From these correlations, there appears to be insufficient reason to continue to use IEAC equations two through five. We know from the winter 2002 study [5] that the calculated result from IEAC₁ is a good estimate of the lower bound for the final cost. Also, it is known with 95 percent confidence that the absolute value of the difference between the CPI at 20 percent complete (CPI₂₀) and the CPI at project completion (CPI₁₀₀) will not be greater than 0.10 [4]. Thus the result from IEAC₁, when using the projected extreme values for the CPI₁₀₀, is expected to yield the upper and lower bounds for the final cost. Only the IEAC₁ equation is needed to predict the range of project cost outcomes with 90 percent confidence². (*A 90 percent confidence for the estimate at completion (EAC) range is equivalent to 95 percent confidence that $|CPI_{20} - CPI_{100}| \leq 0.1$.*)

Alternative Calculation Methods

An alternative to the presently employed IEAC calculation methods (the five formulas cited previously) is to compute the statistical range of outcomes for the CPI. I have described and illustrated this method in a prior publication [6]. Using the range of outcomes for the CPI from the statistical method and IEAC₁, a range for the estimates at completion can be calculated. The range may be computed for any statistical confidence level desired; in my article referenced above, the range is \pm three standard deviations, but it just as well could be 90 percent.

If the CPI for each performance period of the project behaves independently from when it occurs, then this method should yield very good results. Without proof, I believe that the method will still provide reasonable results, even when there is an underlying relationship

between the cumulative value of the CPI and the period of performance in which it occurs. The reason for my assertion is the value of the CPI is updated each period; therefore, it is moving toward its final value. And, the standard deviation of the periodic values is likely large enough to encompass the value for the CPI at project completion. Therefore, it is very likely the actual final cost will be within the 90 percent confidence range calculated using IEAC₁. For this method, the predicted estimate at completion will always be optimistically biased; i.e., it is likely the computed nominal value will be less than the final actual cost.

A second IEAC alternative calculation method is similar to the first, but it should reduce the optimistic bias. The characteristic of the CPI worsens from the point of stability until completion, reported in the 1993 study [3], indicates there may be a mathematical relationship between cumulative CPI and the percentage of project completion: *Beginning at 20 percent complete, the CPI is regarded as stable and proceeds to*

“ ... I believe that the method will still provide reasonable results, even when there is an underlying relationship between the cumulative value of the CPI and the period of performance in which it occurs.”

decrease as percent complete increases, but does not fall more than 0.10 from its stable value.

Having the periodic cumulative values of the CPI indicate increasingly inefficient cost performance as the project nears completion makes intuitive sense. During the early and middle stages of a project there are many tasks for which effort expended will gain earned value. In these stages, if an impediment for a task stands in the way of its accomplishment, there is generally opportunity to do another task. However, if an impediment occurs as the project nears completion, it is highly likely the worker will waste effort until the task can be completed because other tasks are not available. In actuality, what I have

described occurs.

Understanding the stability and inefficiency characteristics, a mathematical model of the CPI decreasing as the project moves toward completion can be created. With the proposed equation, it should be understood that the model only deals with the two characteristics and thus has little theoretical substantiation. The mathematical form chosen, after some experimentation, is the following:

$$\ln \text{CPI} = A + B * (X \wedge 3)$$

where,

A and B are unknown parameters, X is the percentage completion of the project, and ln is the natural logarithm.

The calculated result from the equation is considered to be valid when the project is within the range of 20 percent to 100 percent complete. An advantage of the model's mathematical form is that it has only two unknown parameters: A and B. Parameter A can be either positive or negative; however, B can only be negative. By constraining B to be negative, the tendency of the CPI to worsen from the point of stability to project completion is imposed in the model [4]. The rate of decrease of the CPI is dependent upon B and the power to which X (percent complete) is raised. After some trials, I chose the power 3. It seems reasonable that noticeable efficiency roll-off should begin to occur when X equals 0.5; the power equal to 3 provides this behavior.

Using this model with curve-fitting software, statistical prediction is easily accomplished. The software produces the nominal values for A and B along with their corresponding 90 percent confidence limits.² When applying the curve fitting software, the variables A and B are constrained such that the CPI₁₀₀ is within 0.1 of the CPI₂₀. As mentioned previously, the variable B is further restricted to have only negative values. The constrained values for A and B are computed for each paired data values of the CPI and percent complete using the following equations:

$$\begin{aligned} A_{\text{con}} &= \ln(\text{CPI}_{2n} + 0.1) - B_{\text{max}} \\ B_{\text{con}} &= \ln(\text{CPI}_{2n} - 0.1) - A_{\text{min}} \end{aligned}$$

where,

A_{con} and B_{con} are the constraint values for the variables A and B; A_{min} and B_{max} (= 0) are the minimum and maximum values of A and B, respectively; and the CPI_{2n} is the value occurring at the first percent com-

Percent Complete	The CPI	Percent Complete	The CPI
0.05	1.07896	0.50	1.04279
0.10	1.01511	0.60	1.03634
0.15	1.05731	0.70	1.02942
0.20	1.05961	0.80	1.01076
0.25	1.05844	0.90	0.99260
0.30	1.05211	1.00	0.97287
0.40	1.04970		

Table 1: *Percent Complete and the CPI Data (notional)*

plete greater than 0.2. The value of the CPI_{2n} is assumed to approximate the CPI_{2n} .

Applying the constraints in the curve-fitting software for each new data point maintains proper behavior of the CPI model. The A_{con} constraint is the maximum nominal value of the A variable from the curve fit. Similarly, the variable B is constrained between zero and B_{con} . Because the constraints may affect the curve fit, the nominal value of A should be reviewed for change. If it has changed, use the new value as A_{min} in the B_{con} equation to recalculate the constraint value of B. Enter the computed value for B_{con} and re-perform the curve fit.

The confidence limits produced from the software assume that the data population is infinite. However for our application, project data is finite. For example, the project may execute for two years; if earned value status is taken monthly, the project has 24 data points. Because projects are finite, the confidence limits for ln CPI from the curve fit require adjustment. Multiplying the confidence interval by the following factor will perform the adjustment:

$$\sqrt{((BAC - BCWP) / (BAC - BCWP_{avg}))}$$

where,

$BCWP_{avg}$ is equal to BCWP divided by the number of periodic observations.

The effect of the adjustment factor is to decrease the range for the possible outcomes of the estimates, as the project moves toward completion. For example, at project completion having a range of possible outcomes has no meaning. The adjustment factor reduces the range to zero when the project is finished.

Using the adjusted confidence interval, the 90 percent confidence limits for ln CPI can be computed. Of course, by applying the antilog the upper and lower CPI values are determined. These quantities are then used in $IEAC_1$ to calculate the 90 percent confidence limits of the EAC. The calculation may yield a confidence limit outside of the upper and lower bounds for EAC, especially when percent complete is less than 0.5. In agreement with the studies cited earlier, the lower bound is estimated by dividing the CPI_{2n} plus 0.1 into BAC, while the upper bound is BAC divided by CPI_{2n} minus 0.10.

Example Application

To provide an example of the proposed $IEAC$ calculation method, notional data

has been created for percent complete and the cumulative CPI. The data is shown in Table 1.

Beginning with $X \geq 0.2$, the nominal values of A and B, and their 90 percent confidence limits, are repeatedly obtained from the curve-fit as each new data point is included in the data set. A minimum of three data points is required to determine A and B and their limits. The high and the low values for ln CPI, calculated from the model's formula, are determined by pairing the high values of A and B, and the low values, respectively. The high and low ln CPI limits are then modified by the finite project adjustment factor, as described in the previous section. These adjusted limits are the 90 percent confidence limits for ln CPI.

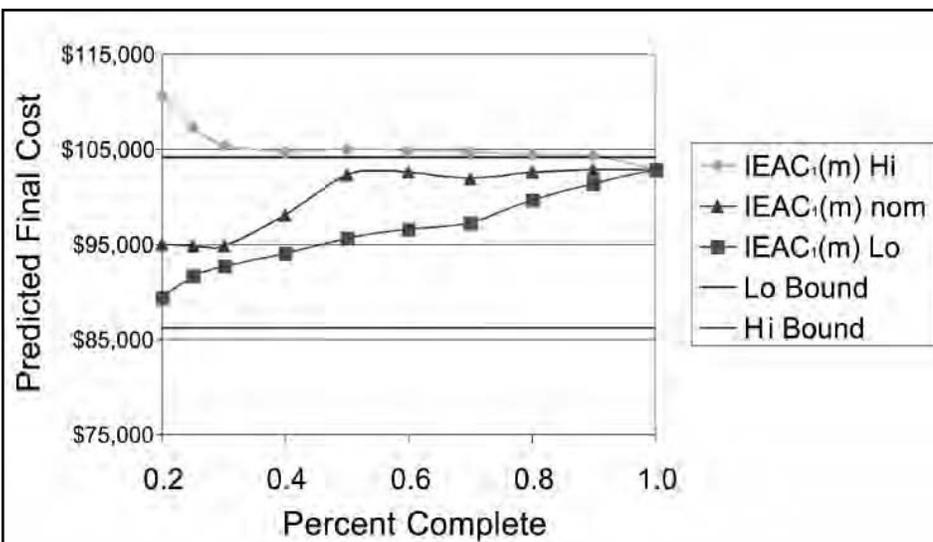
The antilogarithms of the nominal, and the adjusted high and low values, of ln CPI are used in the $IEAC_1$ equation to calculate estimate at completion. The calculation produces the most likely value for EAC along with its 90 percent confidence limits. For the Budget at Completion (BAC) equal to \$100,000, the result of the curve fit for our $IEAC$ model ($IEAC_1(m)$) is shown in Figure 1.

As can be observed, the model rapidly converges and accurately predicts (\$102,817) the final cost (\$102,788), after only a few data points are included in the curve fit. Likewise, it is seen that the 90 percent confidence limits, $IEAC_1(m)$ Hi and $IEAC_1(m)$ Lo, converge and eventually fall within the high and low boundaries for EAC. For reference, the high and low bounds for EAC are shown in the figure as Hi Bound and Lo Bound. The value for the high bound is \$104,209, while the low bound is \$86,236. Thus, it can be said that the results from the $IEAC$ model are well behaved with respect to the predicted extremes of the estimate at completion.

To further illustrate the model's performance, the results from computing $IEAC_1$ and $IEAC_3$ are compared to our model. Recall that $IEAC_3$ uses CPI_3 , which is the cumulative value of the CPI from the last three periodic observations. Figure 2 graphically depicts the percent difference from the final cost for each of the calculation methods. $IEAC_1$ and $IEAC_3$ are calculated using the equations cited earlier, while $IEAC_1(m)$ and $IEAC_1(m)$ Lo are the nominal and low confidence limit values from Figure 1.

As seen from Figure 2, the three methods produce comparably poor results for percent complete equal to 20 percent through 30 percent. Beginning at 40 percent there is a marked departure; the model's prediction of final cost becomes

Figure 1: *IEAC₁ Model*



significantly improved and is better than the other estimates. Beginning at 50 percent and continuing through project completion, the method used to calculate $IEAC_i(m)$ produces cost estimates that have very small differences with the actual final cost. It is also noticeable that $IEAC_i$ provides optimistic results as it should if, indeed, the CPI tends to worsen as percent complete increases [3, 4]. Likewise, $IEAC_s$ produces optimistic results as percent complete increases, again, due to the tendency of the CPI to worsen. While not observed in the example, the CPI model can produce either optimistic or pessimistic results for $IEAC_i(m)$.

The final observation for Figure 2 is the comparison of $IEAC_i$ to $IEAC_i(m)$ Lo. Recall from the earlier discussion in the studies section of this article that $IEAC_i$ was postulated to provide a good running estimate for the lowest value for final cost. The figure shows the two lines closely tracking beginning at percent complete equal to 0.40. Thus if the hypothesis concerning $IEAC_i$ is valid, there is added credence to the nominal and high confidence limit values produced by the $IEAC_i(m)$ calculation method.

Although the graphical result appears wonderful, the method is unproven. The data created conforms to the model itself; thus, the result should be good. Even so, what has been shown is significant. The model presented for the CPI behaves in accordance with the behavior characteristics determined by previous studies [3, 4, 5]:

1. The CPI stabilizes when project reaches 20 percent complete, CPI_{20} .
2. The CPI tends only to worsen from the point of stability until project completion.
3. With 95 percent confidence, the CPI at project completion will not be more than 0.10 from CPI_{20} .

If these characteristics are indeed true, then our example indicates the proposed model may provide good prediction capability for estimate at completion.

Prototype Application

To further illustrate the curve fit model approach to calculating $IEAC$, results from prototyping actual project data are shown. The application is in progress. As can be seen, the results from the real data correlate well with the notional data presented earlier. Figure 3 is an output from the curve fit software. In agreement with the studies of the CPI, the $\ln CPI$ is seen worsening as percent complete increases. As discussed earlier, the model accounts for degradation of cost performance as

the project nears completion. Figure 4 illustrates the model's rapid convergence to the predicted final cost. Observed in Figure 5 (see page 30), both $IEAC_i$ and $IEAC_s$ are predicting a more optimistic final cost than is the model. Lastly, the close tracking of $IEAC_i(m)$ Lo $IEAC_i$ is

strikingly similar to the observation made for the notional data.

Summary

From several previous studies, it can be inferred that the behavior of the CPI, from its point of stability to project com-

Figure 2: $IEAC$ Comparison

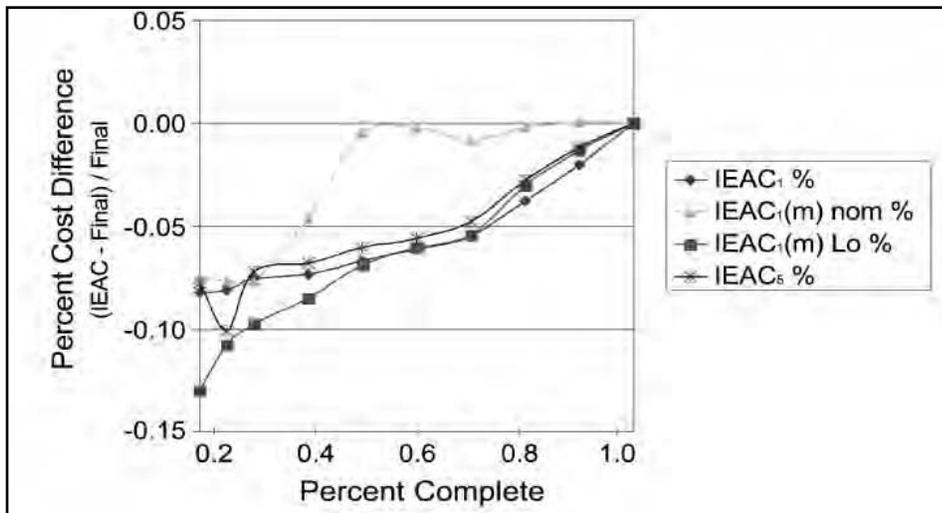


Figure 3: $\ln CPI$ Versus Percent Complete (Prototype)

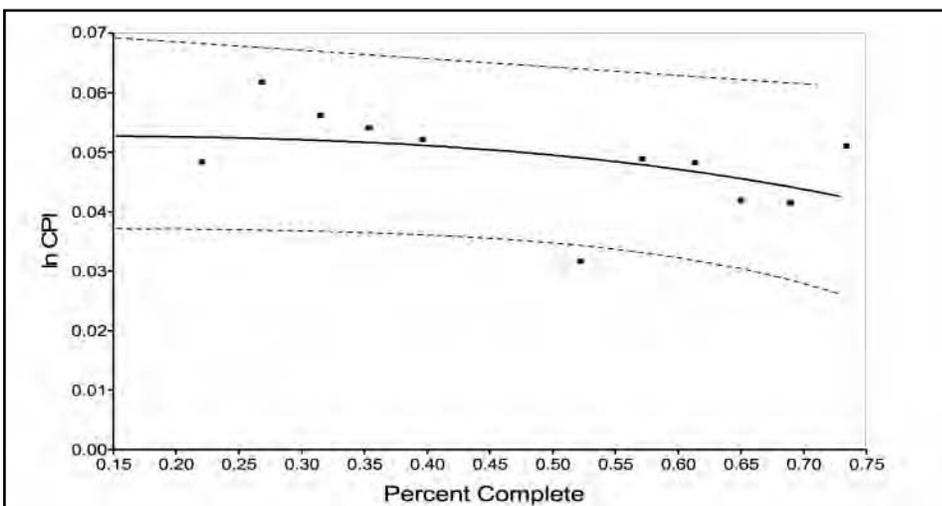
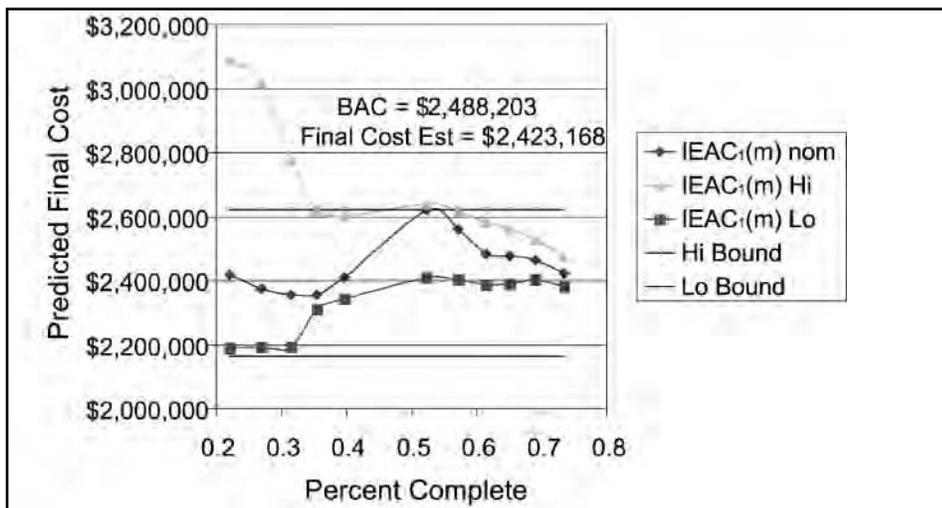


Figure 4: $IEAC_i$ Analysis (Prototype)



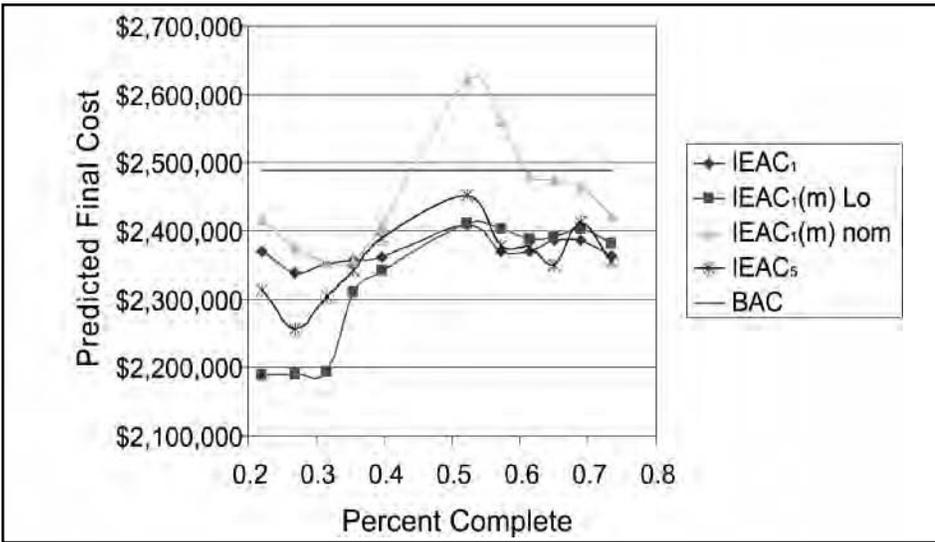


Figure 5: IEAC Comparison (Prototype)

pletion, explains why the various formulations of IEAC yield reasonable results for specific conditions. By incorporating this behavior into a mathematical model for the CPI, it is proposed the only formula needed for estimating the cost at completion is IEAC_i; the cumulative value of the CPI is replaced in the formula by the value from the model.

The model for the CPI is constructed to behave in accordance with characteristics determined by past studies. The values of the CPI will tend to decrease as the completion percentage of the project increases. The amount of decrease is constrained to agree with the statistical testing studies.

Using the mathematical model for the CPI with curve-fitting software and associated statistical methods, the independent estimate at completion with its 90 percent confidence limits can be computed. As indicated from the application to both notional and real data, the proposed method yields excellent results. The method may be an improvement to the IEAC equations presently applied.◆

Recommendation

To validate for general application, the CPI behavior model and IEAC calculation method discussed in this article should have an independent study performed using data from the Department of Defense earned value database.

References

1. Fleming, Q. Cost/Schedule Control Systems Criteria, The Management Guide to C/SCSC. Probus, 1988.
2. Christensen, D.S. "The Estimate at Completion Problem: A Review of Three Studies." Project Management Journal Vol. 24 (Mar. 1993): 37-42.
3. Christensen, D.S., R.C. Antolini, and

J.W. McKinney. "A Review of Estimate at Completion Research." Journal of Cost Analysis and Management Spring 1995: 41-62.

4. Christensen, D.S., S.R. Heise. "Cost Performance Index Stability." National Contract Management Journal Vol. 25 (1993): 7-15.
5. Christensen, D.S., and C. Templin. "EAC Evaluation Methods: Do They Still Work?" Acquisition Review Quarterly Spring 2002: 105-116.
6. Christensen, D.S., and D.A. Rees. "Is the CPI-Based EAC a Lower Bound to the Final Cost of Post A-12 Contracts?" Journal of Cost Analysis and Management Winter 2002: 55-65.
7. Lipke, W. "The Probability of Success." CROSSTALK, Nov. 2003 <www.stsc.hill.af.mil/crosstalk/2003/11/0311Lipke.html>.
8. Crow, E.L., F.A. Davis, and M.W. Maxfield. Statistics Manual. New York: Dover Publications, 1960.

Notes

1. The definitions of the cost and schedule performance indexes (the CPI and SPI, respectively), and cost variance (CV) are as follows:

$$\begin{aligned} \text{The CPI} &= \text{BCWP} / \text{ACWP} \\ \text{SPI} &= \text{BCWP} / \text{BCWS} \\ \text{CV} &= \text{BCWP} - \text{ACWP} \end{aligned}$$

where,

$$\begin{aligned} \text{ACWP} &= \text{Actual Cost for Work Performed} \\ \text{BCWP} &= \text{Budgeted Cost for Work Performed (earned value)} \\ \text{BCWS} &= \text{Budgeted Cost for Work Scheduled (project performance baseline)} \end{aligned}$$

- For a more in-depth explanation of earned value and its indicators, reference Quentin Fleming's book [7].
2. The Confidence Interval is the region surrounding the computed nominal value within which the true value lies with a specified level of confidence. The end points of the interval are the Confidence Limits. The equation for the Confidence Limits is:

$$\langle x \rangle \pm z (\sigma / n)$$

where,

$\langle x \rangle$ is the nominal value of x, while z is from the standard unit normal distribution and corresponds to the area selected (for this application, z = 1.6449 at 90 percent of the distribution area), σ is the standard deviation of the observations of x, and n is the number of observations [8].

About the Author



Walt Lipke is the deputy chief of the Software Division at the Oklahoma City Air Logistics Center. He has 30 years of experience in the

development, maintenance, and management of software for automated testing of avionics. In 1993 with his guidance, the Test Program Set and Industrial Automation (TPS and IA) functions of the division became the first Air Force activity to achieve Level 2 of the Software Engineering Institute's Capability Maturity Model® (CMM®). In 1996, these functions became the first software activity in federal service to achieve CMM Level 4 distinction. Under Lipke's direction, the TPS and IA functions became ISO 9001/TickIT registered in 1998. These same functions were honored in 1999 with the Institute of Electrical and Electronics Engineers' Computer Society Award for Software Process Achievement. Lipke is a professional engineer with a master's degree in physics.

OC-ALC/MAS
Tinker AFB, OK 73145-9144
Phone: (405) 736-3341
Fax: (405) 736-3345
E-mail: walter.lipke@tinker.af.mil



A little Learning is a dang'rous Thing; Drink deep, or taste not the Pierian Spring¹.

— Alexander Pope²

While many people are familiar with metadata, which is information about information, and widely used to manage content in cyberspace, few understand the realm of *pseudo-knowledge*, where a little learning is not only not a dangerous thing, but in fact constitutes the most effective form of just-in-time transfer of intellectual property ever devised. Consider the pseudo-knowledge for the Capability Maturity Model[®] (CMM)³ that was carefully crafted by its authors nearly two decades ago (whether they knew it or not), supporting countless presentations to senior managers in which software engineering process group chairs want to familiarize them with the CMM, but don't want them to know enough to be dangerous:

1. The CMM has five maturity levels (so you can count them on one hand, boss).
2. Those five maturity levels have 18 key process areas (just like the 18 holes on the golf course you're going to play this afternoon with that potential client).
3. Those 18 key process areas have 52 goals to achieve (just as there are 52 cards in the deck you'll use to play poker at the 19th hole, after your 18-hole round of golf).
4. Those 52 goals are satisfied through the implementation of 316 key practices (which is almost 317, or if presented as 3/17 would be recognized as March 17th, which as we all know is St. Patrick's Day, and a fine representative for the beer to be consumed at the 19th hole while you play poker with a deck of 52 cards after finishing the 18 holes of golf and shaking hands with your now new client with that firm senior manager handshake using all five fingers! However, since we only have 316 key practices, we'll just have to remember St. Patrick's Day eve, which would then of course be March 16th, or 3/16).

Thus, through effective presentation of pseudo-knowledge, your senior manager has information relevant to the CMM with which he or she can amuse his or her peers, family, and friends at cocktail parties without being dangerous to you, the process improvement lead!

Now, however, with the transition to the CMM IntegrationSM (CMMI[®]), this treasure trove of pseudo-knowledge is soon to be rendered obsolete! Useless! Pointless! The question of the hour is this: What shall be the pseudo-knowledge associated with the

CMMI? How is a poor process lead to portray relevant but useless information about the CMMI to management, friends, family, and neighbors? Where do you start? Staged, continuous, or constageduous? Software only? Software and systems engineering? How about Integrated Product and Process Development (IPPD) and supplier sourcing? Six capability levels or five maturity levels? Where to begin? While it is true that each representation has 25 process areas and 55 specific goals, just what do you do with all those generic goals and practices? And how do you account for the difference in specific practices (189 vs. 185)? Do you even attempt to address the bodies of knowledge incorporated, the different dimensions, or the categories of process areas? Inquiring minds want to know!

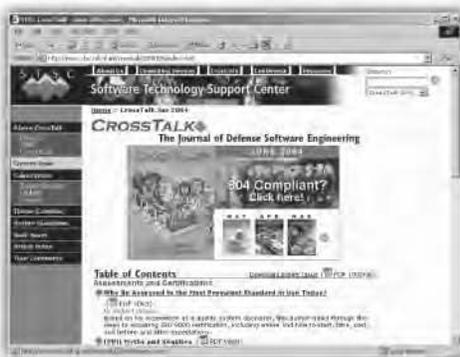
Earlier this year, Pat O'Toole calculated that there are 4.7 x 10¹⁸ possible capability level profiles across all 24 process areas in the CMMI for Systems Engineering, Software Engineering and IPPD Vers. 1.1 since each may be performed at any one of the six possible capability levels⁴. How exactly do you think we'll capture that little tidbit in pseudo-knowledge?

To this end, I hope the process improvement community will accept the challenge to identify the new pseudo-knowledge for the CMMI. How should we represent this fascinating tool in all its glory and splendor without sharing anything of real meaning or value? How shall we endeavor to entertain and amuse without compromising our positions as organizational leaders of process improvement? Put on your pseudo-thinking caps and send me your best! We'll select the finalists from all entries submitted to me by Oct. 29, and let the community pick the winners at the CMMI conference in Denver, Colo., Nov. 15-18.

— Barry Schrimsher
barry@glentalon.com

1. The American Heritage[®] Dictionary of the English Language: Fourth Edition. 2000. Pierian Spring NOUN: 1. *Greek Mythology* A spring in Macedonia, sacred to the Muses. 2. A source of inspiration.
2. Pope, Alexander. *Poetry and Prose of Alexander Pope*. Houghton Mifflin Co., 1 June 1969.
3. Adapted with permission from a tale shared by Pat O'Toole of Process Assessment, Consulting, & Training, LLC.
4. O'Toole, Pat. "Do's and Don'ts of Process Improvement, #18: Don't Maintain a Low Profile." 3 Apr. 2004.

CROSSTALK ARCHIVES



Visit CROSSTALK online and access 11 years of software-related articles in a fully searchable database that makes finding relevant and timely information painless. Make CROSSTALK your first stop for all your software research needs.

www.stsc.hill.af.mil/crosstalk

PROCESS IMPROVEMENT

g e t t i n g
Y O U R
t e a m
i n l i n e



The *Software Technology Support Center* (STSC) helps you improve your processes by identifying root causes of your problems; constructing simple, practical solutions; and creating ownership of the solution. In other words, we help determine what to change, what to change to, and how to cause the change. But how do you get started?

Call us. The STSC can help plan process improvement at any level. Once you decide what you need, we are the group to help implement the solution, whether it's implementing the Capability Maturity

Model®, setting up a Management Steering Group, or designing a Software Engineering Process Group infrastructure. We help you understand how to implement process improvement.

Don't start your process improvement without a STSC mentor.

We have been in the trenches and have the scars to prove it. Call us first. Whether your organization is big or small, just starting a project or embattled in difficulties, we can help. We bring hands-on experience.

OO-ALC/MAS • 6022 Fir Avenue • Building 1238 • Hill AFB, UT 84056-5820 • 801 775 5555 • www.stsc.hill.af.mil



Co-Sponsored by
U.S. Air Force
Air Logistics Centers
MAS Software Divisions

CROSSTALK / MASE

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

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