CROSSTALK

April 2000 The Journal of Defense Software Engineering Vol. 13 No. 4

COST ESTIMATION

Cost Estimation

On the Cover:
Cover artist Mark Driscoll is a designer and technical illustrator with a background in architecture and museum exhibit design. He is one of the principals of Driscoll Design Inc. in Salt Lake City.

Software Engineering Technology

Open Forum

Departments

CrossTalk

Sponsor	H. Bruce Allgood
Publisher	Reuel S. Alder
Associate Publisher	Lynn Silver
Managing Editor	Kathy Gurchiek
Associate Editor/Layout	Matthew Welker
Associate Editor/Features	Heather Winward
Voice	801-775-5555
Fax	801-777-8069
E-mail	crosstalk.staff@hill.af.mil
STSC Online	http://www.stsc.hill.af.mil
CrossTalk Online	http://www.stsc.hill.af.mil/Crosstalk/crostalk.html
CRSIP Online	http://www.crsip.hill.af.mil

Subscriptions : Send correspondence concerning subscriptions and changes of address to the following address. You may use the form on page 31.
Ogden ALC/TISE
7278 Fourth Street
Hill AFB, Utah 84056-5205

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

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

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

Coming Events : We often list conferences, seminars, symposiums, etc., that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CrossTalk Editorial Department.
STSC Online Services: at http://www.stsc.hill.af.mil.
Call 801-777-7026, e-mail randy.schreifels@hill.af.mil.

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

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Making an Educated Guess

What does estimation mean? Having spent 30 years as an engineer and manager in the electronics and software marketplace, I thought, "Oh, that's easy! You simply *guess* how long it is going to take you to do this job, based on your last experience with the same or a similar job, and then *double* that guess."

Two things are clear with this oft-used algorithm. The first is that to have any kind of chance to be in the ballpark, you need some knowledge or historical precedence for having done something similar in the not-too-distant past. The second is a realization that the accuracy of any such estimate is not very good.

If your business success does not rely on the accuracy of this type of estimate, this algorithm will probably continue to be used. Only when a business' existence and success rely on something a little more accurate is it likely a little more effort will be applied to make cost estimates more realistic and reliable.

As readers will find in this month's *CROSSTALK,* cost estimation has become an essential feature of any successful software business development or sustainment venture. However, as quoted in *Reducing Bias in Software Project Estimates* by David Peeters and George Dewey on page 20, some still believe that "software estimating is definitely a black art." The authors note that a large percentage of software projects continue to finish behind schedule and over budget. The article discusses some ways to identify and reduce biases in software cost estimation to help make estimates more accurate.

The dramatic increase in the quantity and complexity of software that drives so many of today's defense and consumer products is also a major driver in the need to develop more robust estimation methodologies and tools. Barry Boehm and his co-authors write in *Future Trends, Implications in Software Cost Estimation Models* on page 4 that software development trends have turned from the Waterfall process model toward evolutionary, incremental, and spiral models. It was found that the very useful Constructive Cost Model (COCOMO) estimation techniques required some significant changes to keep pace with the changing nature of this software development evolution. Product line management approaches to software reuse, and graphic user interface builder tools that made traditional-size metrics such as source lines of code inappropriate are examples of new software design methods that required new techniques. This article also discusses how continuing development extensions are being made to COCOMO II to address emerging trends such as Rapid Application Development and commercial-off-the-shelf integration.

We hope that readers will find these and the other articles in this month's *CROSSTALK* to be valuable resources as they continually improve their organization's ability to accurately predict project schedule and cost.

H. Bruce Allgood
Deputy Computer Resources Support Improvement Program Director

CROSSTALK welcomes Bruce Allgood, Deputy of the Computer Resources Support Improvement Program (CRSIP) at Hill Air Force Base, Utah. Allgood replaces Lt. Col. Joseph Jarzombek (retired) as the CRSIP sponsor of our journal. As a member of the Air Force Software Technology Support Center, Allgood has supported software process improvement efforts throughout the Air Force and the Department of Defense. He also represents the Air Force Materiel Command on the Practical Software Measurement Technical Steering Group (PSM), the Office of the Secretary of Defense's Software Collaboration Team, and is a certified PSM trainer. He spent 20 years in various management and development roles at leading commercial electronics and software corporations, including 11 years at IBM, two years at Hughes Aircraft, and five years at Hewlett Packard. Allgood received his bachelor's degree in electrical engineering from the University of Utah and a master's degree in electrical engineering from Colorado State University.

Future Trends, Implications in Cost Estimation Models

The rapid pace of change in software technology requires everybody in the software business to continually rethink and update their practices just to stay relevant and effective. This article discusses this challenge first with respect to the USC COCOMO II software cost modeling project, and then for software-intensive organizations in general. It then presents a series of adaptive feedback loops by which organizations can use COCOMO II-type models to help cope with the challenges of change.

A major team effort was recently completed to re-engineer the original Constructive Cost Model (COCOMO) for software cost and schedule estimation into a new model, COCOMO II. The overall COCOMO framework remained about the same, but significant changes were found to be necessary to keep pace with the changing nature of software development and evolution. These trends have included a move away from the Waterfall process model toward evolutionary, incremental, and spiral models; product line management approaches to software reuse; applications composition capabilities; and graphic user interface builder tools that made traditional size metrics such as source lines of code (SLOC) inappropriate.

We have replaced the COCOMO development modes (organic, semidetached, embedded) by a set of scale factors (precedentedness, development flexibility, architecture and risk resolution, team cohesiveness, and process maturity). These enable project managers to control that which affects their project's economies and diseconomies of scale. We added some multiplicative cost drivers (development for reuse, degree of documentation, multisite development); dropped the Turnaround Time cost driver; merged the Modern Programming Practices cost driver into the process maturity scale factor, and changed the requirements volatility cost driver into a size factor.

We changed the main size parameter from Delivered Source Instructions to a user-determined mix of SLOC and function points; changed to a more detailed nonlinear model of software reuse effects; and provided a family of models (Applications Composition, Early Design, and Post-Architecture) tuned to the information available at different stages of the development process. We developed a Bayesian approach to calibration of COCOMO II to 161 projects from 18 organizations, resulting in a model that estimates within 30 percent of the actual effort 75 percent of the time (80 percent of the time if calibrated to the individual organizations' data). A book describing the model is to be released in June; it will include a CD with a USC COCOMO II tool and demo versions of three commercial implementations [1]. Further information about COCOMO II is available at http://sunset.usc.edu/COCOMOII/suite.html

Rather than leaving the model as is for the next 18 years as with the original COCOMO, we are determining extensions to COCOMO II to address emerging trends such as Rapid Application Development (RAD) and commercial off-the-shelf (COTS) integration. This need to continually update your software estimation capabilities also holds for most organizations. This paper explores the reasons for this and some of the implications.

Trends in Software Productivity, Estimating Accuracy

In principle, your organization should be able to continuously measure, recalibrate, and refine models such as COCOMO II to converge uniformly toward perfection in understanding your software applications and in accurately estimating the costs and schedules.

In practice, convergence toward perfection in estimation is not likely to be uniform. Two major phenomena are likely to interrupt your progress in estimation accuracy:

1. As your understanding increases about the nature of your applications domain, you will also be able to improve your software productivity and quality by using larger solution components and more powerful applications definition languages. Changing to these construction methods will require you to revise your estimation techniques, and will cause your estimation error to increase.
2. The overall pace of change via new technologies and paradigm shifts in the nature of software products, processes, organizations, and people will cause the inputs and outputs of software estimation models to change. Again, these changes are likely to improve software productivity and quality, but cause your estimation error to increase.

Effects of Increasing Domain Understanding

Suppose you are entering a new applications domain, (e.g., control of distributed, heterogeneous, real-time automated agents for robotics devices). Your initial software productivity in this domain is likely to be low, largely due to the effects of such COCOMO II variables as precendentedness, architecture and risk resolution, complexity, and applications experience. In particular, your understanding of the architecture for such systems and your ability to reuse components will be low. And your unfamiliarity with the domain will cause your cost and schedule estimation errors to be relatively high.

As you increase your understanding of how to build such systems and their components, both your productivity and your estimation accuracy will improve. However, at some point you will understand enough about the domain to begin developing a product line architecture and reusable components to be used in future products. At this point (point A in Figure 1), your productivity will go up faster, as you will be reusing rather than developing more and more of the software (in COCOMO II terms, your equivalent SLOC will decrease for the same type of project). However, at point A, your estimation error will go up, as your previous cost driver ratings will be less relevant, and you will be starting on the learning curve in rating your reuse parameters. You will also find that reuse and product line management cause significant changes in your processes [2, 3].

As you improve your understanding of how to increase productivity and reduce estimation error in using component-based development, you will often find that other organizations in the domain are doing so as well. Soon, some of the more general

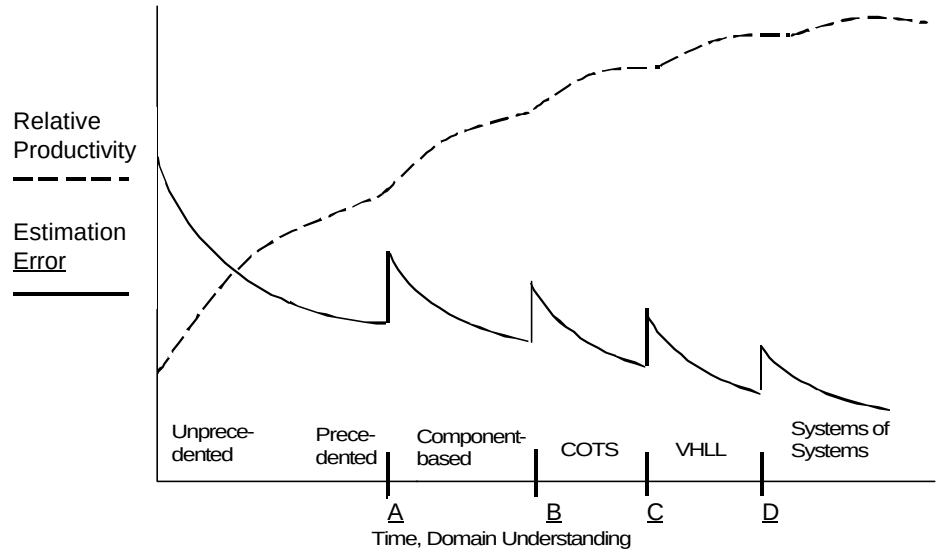


Figure 1. *Productivity and Estimation Accuracy Trends*

components will be shared across organizations or offered as COTS products. With their development and maintenance costs amortized over more and more user organizations, they become cheaper to employ than some of your reusable components. Again, using these COTS products or shared components will increase your productivity rate (point B in Figure 1). Initially, you will find it harder to predict the cost and schedule of integrating heterogeneous COTS components with your components and with each other, and your estimation error at point B will also go up.

VHLL's and System of Systems

This scenario will generally repeat itself at points C and D in Figure 1. At point C, you and/or others will know enough about how to compose domain components to be able to automate their composition, and to provide a domain-specific Very High Level Language (VHLL) with user-oriented terminology to specify the particular application desired. Productivity rates will increase (in COCOMO II terms, via the need for much fewer source lines of code), but estimation errors also initially go up.

At point D, you will find that there is a demand to closely integrate your VHLL-driven robotic device systems for subassembly manufacturing, for example, with other VHLL's and application generators for factory control systems and electronic commerce systems, into a total factory system of systems. Integrating the systems will be more productive than building a whole new factory system, but your error in estimating cost and schedule will be higher than for an individual system. This is because of uncertainties you will have in estimating the effort required to reconcile the unpredictable incompatibilities in interfaces, priorities, assumptions, and usage conventions among the subassembly manufacturing, factory control, and electronic commerce VHLL's and systems [4].

Effects of Innovation, Change

Other sources of innovation and change may cause changes in the nature of your software projects' product, process, organization, and people. These may improve your organization's overall productivity, but their effect on your projects' practice may again increase your estimation error.

In the area of product technology, such changes have included changes from batch-processing to interactive systems, and from single mainframes to distributed and networked systems. Other product technologies such as graphic user interface builders will also increase productivity, but estimation error increases because of new challenges in determining what to count as product size.

In the area of process technology, the change from waterfall to evolutionary or spiral development requires rethinking the project's endpoints and phases. Incremental development, RAD, cost-as-independent-variable (CAIV), or schedule-as-independent-variable (SAIV) cause further rethinking of process strategies, endpoints, and phases. With CAIV or SAIV, for example, you may specify and design more product than you deliver when you run out of budget or schedule. Collaborative processes (Joint Application Development, Integrated Product Team, etc.) require involving users, operators, and others in product definition. Should their effort be included in the estimate? To what extent will virtual-reality distributed collaboration technology improve software costs and schedules?

With organizations and people, changes in organizational objectives affect products, processes, and estimation accuracy. One example is the increasing emphasis on reducing schedule (time to market) in order to remain competitive, rather than minimizing cost. Another example is the effect of increasing emphasis on software quality as a competitive discriminator.

The effects of having tens of millions of computer-literate people will also change the nature of software products and processes. Also, the increasingly critical nature of software to an organization's competitive success creates stronger needs for integrating software estimates into business-case and financial-performance models. Trends toward human economics will affect both software products' required functions and user interfaces.

Estimation Accuracy: The Bottom Line

If only our software engineering domain understanding, product and process technology, and organization and people factors stayed constant, we could get uniformly better and better at estimating. But they do not stay constant, and their changes are generally good for people and organizations. The need to continually rethink and re-engineer our software estimation models is a necessary price to pay for the ability to incorporate software engineering improvements.

Coping with Change: COCOMO II

We are trying to ensure that COCOMO II will be adaptive to change by trying to anticipate trends in software engineering practice, as discussed in the Introduction. The resulting three-stage set of COCOMO II models (application composition, early design, post-architecture) anticipates some dimensions of

future change. Other dimensions are addressed by the new or extended cost drivers such as process maturity, architecture and risk resolution, team cohesion, multisite development, use of tools, and the various reuse parameters.

We are also attempting to anticipate future trends via our overall Model-Based (System) Architecting and Software Engineering (MBASE) project. MBASE's key objective is to avoid harmful model clashes by integrating a project's product, process, property, and success models [5]. The COCOMO II suite of models is our main effort in the property model area. Concurrently, we are integrating complementary research into product models (domain, requirements, and architecture models); process models (WinWin spiral model, process anchor points); and success models (stakeholder win-win, business case analysis, *I will know it when I see it* prototyping).

We have been trying to understand and anticipate trends in software engineering product, process, property, and success models via workshops with our affiliates, via model research, and via model experimentation with our annual series of digital library applications projects using MBASE [6]. For example, our initial formulation of the Constructive COTS Integration Cost Model (COCOTS) was based on an affiliates' workshop on COTS integration, and our efforts to incorporate COTS assessment and integration into MBASE extensions of spiral process models and object-oriented product models. Our major refinement of COCOTS into a family of four models was based on analysis of COTS integration experience data from the MBASE digital library projects.

Similarly, our formulation of the Constructive Rapid Application Development Estimation Model (CORADMO) has been based on an affiliates' RAD workshop, and on integrating RAD process models such as schedule-as-independent-variable (SAIV) into MBASE. This was done via RAD experimentation using the digital library projects. These projects are good RAD examples, as our semester constraints require them to be fully architected in 11 weeks, and fully developed and transitioned in another 12 weeks.

Thus, the emerging extensions of COCOMO II discussed in Chapter 5 of *Software Cost Estimation with COCOMO II* (COCOTS, CORADMO, Applications Composition, and other models) represent hypotheses of how to model the cost, schedule, and quality effects of current and future trends in software engineering practice. As we gather more data, we will be able to test and refine these models, and to identify further models or extensions likely to be important for future software engineering practice.

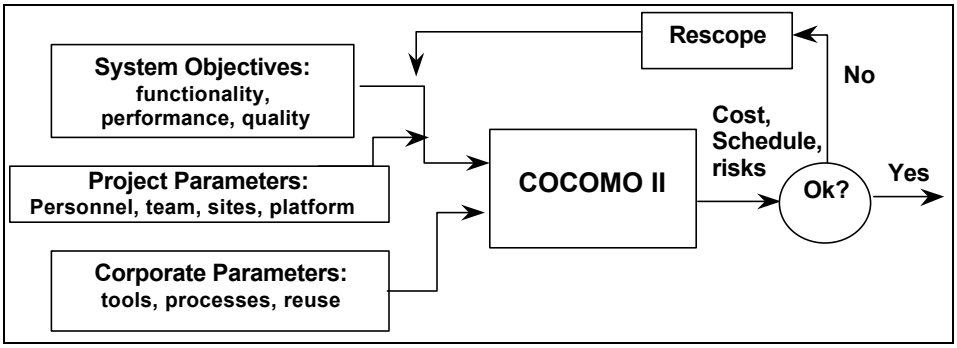


Figure 2. *Using COCOMO II to Cope With Change*

Coping with Change: COCOMO II and Your Organization

COCOMO II can be a useful tool for your organization to use in adapting to future change, both at the project level and at the organizational level.

Coping with Change During Project Definition

Figure 2 shows how COCOMO II can be used to help address issues of change at the project definition level. You can enter your organization's customary values via the COCOMO II parameters, and indicate which ones will undergo change. COCOMO II will estimate how these changes will affect the project's expected cost and schedule, and will provide you and your stakeholders with a framework for rescoping the project if estimated cost and schedule are unsatisfactory.

Coping with Change During Project Execution

Frequently, changes in project objectives, priorities, available componentry, or personnel occur during project execution. If these are anticipated, COCOMO II can support a variant of the project definition process above to converge on a stakeholder-satisfactory rescoping of the project.

A more serious case occurs when the changes are unanticipated and largely unnoticed: via personnel changes; COTS product, reusable component, or tool shortfalls; requirements creep; or platform discontinuities. In such cases, COCOMO II phase and activity distributions can be used to develop a quantitative milestone plan or an earned-value system [7] for the project, which enable plan deviations to be detected, and appropriate corrective actions to be taken (Figure 3) involving COCOMO II in project rescoping.

Coping with Required COCOMO II Model Changes

At times, unanticipated project changes are indications that your COCOMO II model needs to be recalibrated or extended. The more management data you collect on actual project costs and schedules, the better you will be able to do this (see Figure 4).

Recalibration might be appropriate, for example, if your organization is acquired by or merged into an organization with different definitions of project endpoints, or with different definitions of which types of employees are directly charged to the project vs. being changed to overhead. As described in Chapter 4 of *Software Cost Estimation with COCOMO II*, techniques are available to recalibrate COCOMO II's base coefficients and exponents for cost and schedule estimation. Some COCOMO II tools such as USC COCOMO II and COSTAR, a commercial product from SoftStar Systems, provide such calibration features.

Extending the model will be appropriate if some factor assumed to be constant or insignificant turns out to be a significant cost driver. For example, the COCOMO 81 TOOL Factor was not in the original 1978 TRW version of COCOMO, as previous TRW projects had operated with a relatively uniform set of mainframe tools. The TOOL Factor

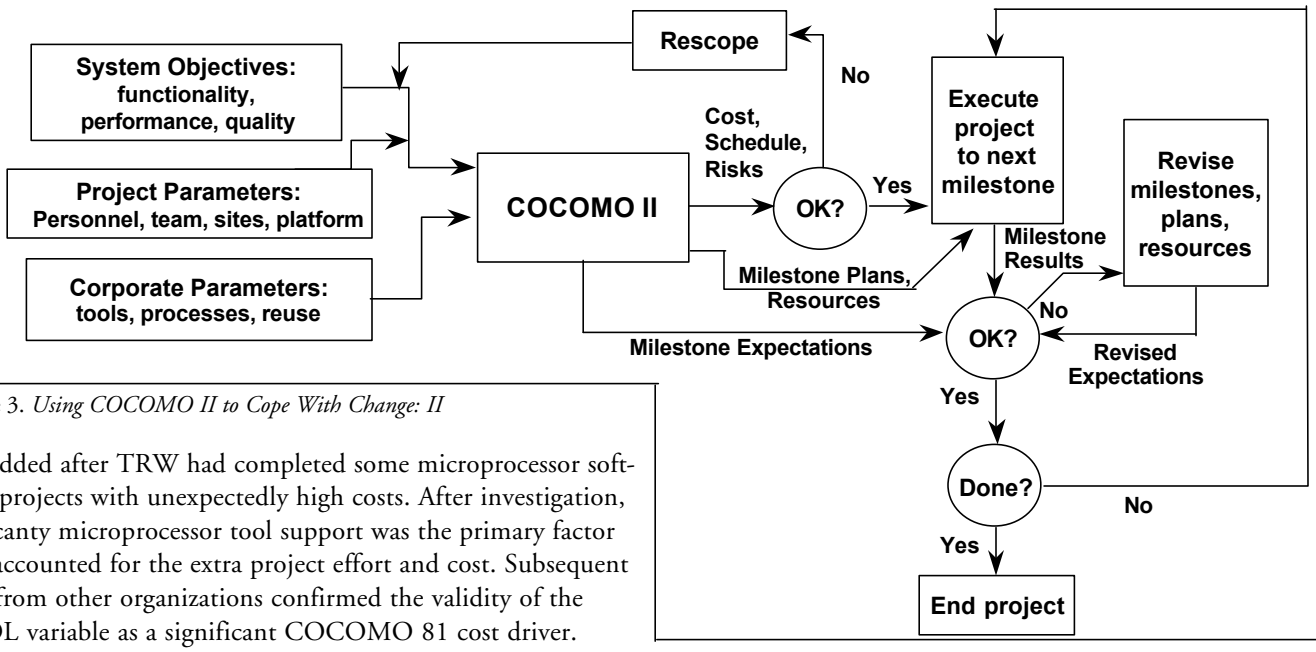


Figure 3. *Using COCOMO II to Cope With Change: II*

was added after TRW had completed some microprocessor software projects with unexpectedly high costs. After investigation, the scanty microprocessor tool support was the primary factor that accounted for the extra project effort and cost. Subsequent data from other organizations confirmed the validity of the TOOL variable as a significant COCOMO 81 cost driver.

Similarly, several variables were added to COCOMO 81 to produce COCOMO II, in response to affiliate indications of need and our confirmation via behavioral analysis.

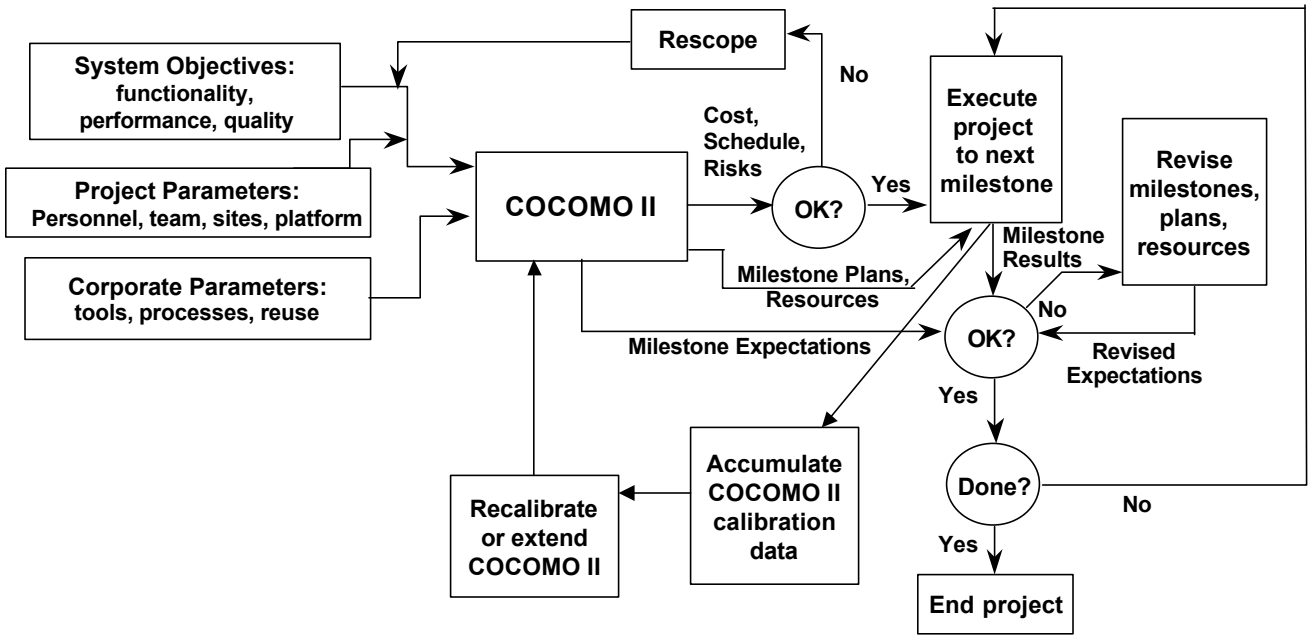
Proactive Organizational Change Management

Your organization will be much better off once it moves away from reacting to change, and toward proactive anticipation and management of change. This is what Level 5 of the SEI-CMM® is all about, particularly the key process areas of Technical Change Management and Process Change Management.

The COCOMO II model and parameters can help you evaluate candidate change management strategies. For example, investing in sufficient software tool acquisition and training to bring your projects' TOOL rating from nominal to high will replace a 1.0 effort multiplier by an 0.90, for a 10 percent productivity gain. Similar investments in improving process maturity, architecture and risk resolution, team cohesion, multisite development, reuse, or any of the personnel factors can also have significant benefits that can be investigated via COCOMO II (See Figure 5). The cost, schedule, and quality drivers of COCOTS and CORADMO can be used similarly.

An integrated capability for using COCOMO II and CORADMO for evaluating the payoff of cost and schedule improvement strategies is provided by the Constructive Productivity Model (COPROMO) extension described in *Software Cost Estimation with COCOMO II*. It enables you to start from a current baseline of cost and schedule drivers from either your own organization's data or the COCOMO II database; and to express candidate cost and schedule improvement strategies in terms of achievable time-phased improvements in cost and schedule drivers. COPROMO will generate the resulting estimates and provide time histories of cost and schedule improvements for each of the candidate strategies.

Figure 4. *Using COCOMO II to Cope With Change: III*



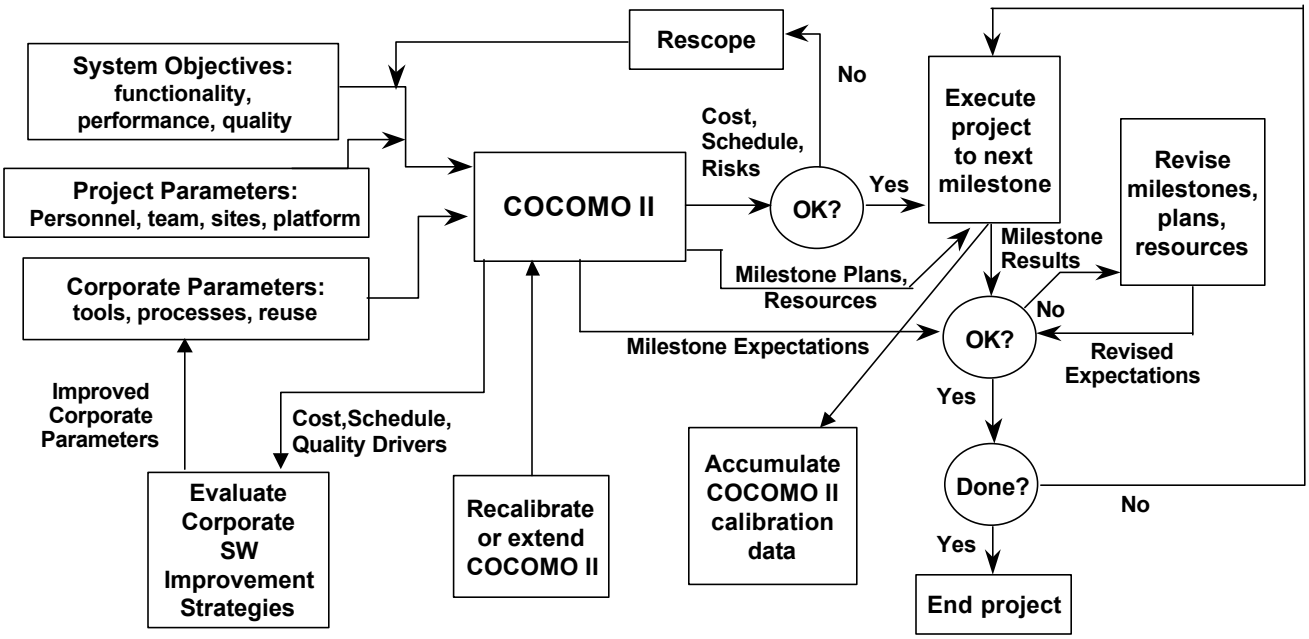


Figure 5. *Using COCOMO II to Cope With Change: IV*

Put together, the four COCOMO II feedback cycles in Figure 5 can enable your organization to determine and evolve a project-level and organization-level set of project analysis, management, and improvement strategies based on your own quantitative metrics. These strategies will enable you to determine appropriate objectives and approaches for each project, to manage projects to more successful completion, and to improve your organization's software productivity, speed, and quality by anticipating and capitalizing on change rather than being a reactive victim of change.

References

1. Boehm, B.; Abts, A.; Brown, W.; Chulani, S.; Clark, B.; Horowitz, E.; Madachy R.; Reifer, D.; and Steece, B. *Software Cost Estimation with COCOMO II,* Prentice Hall (to appear in June 2000).
2. Boehm, B.; Kellner, M. and Perry, D. (eds.), *Proceedings, ISPW 10: Process Support of Software Product Lines,* IEEE Computer Society, 1998.
3. Reifer, D. *Practical Software Reuse,* John Wiley and Sons, 1997.
4. Maier, M. Architecting Principles for Systems-of-Systems, *Systems Engineering* Vol. 1 No. 4 (1998), pp. 267-284.
5. Boehm, B. and Port, D. Escaping the Software Tar Pit: Model Clashes and How to Avoid Them, *ACM Software Engineering Notes,* Jan. 1999, pp. 36-48.
6. Boehm, B.; Egyed, A.; Port, D.; Shah, A.; Kwan, J.; and Madachy R., A Stakeholder Win-Win Approach to Software Engineering Education, *Annals of Software Engineering,* Vol. 6(1998), pp. 295-321.

About the Authors

Barry Boehm is the TRW Professor of Software Engineering and Director of the Center for Software Engineering at USC. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, and the Office of the Secretary of Defense as the Director of Defense Research & Engineering Software and Computer Technology Office. Besides COCOMO, he originated the spiral model of the software process and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
Los Angeles, Calif. 90089-0781
Voice: 213-740-8163
Fax: 213-740-4927
E-mail: boehm@sunset.usc.edu

Dr. Ellis Horowitz is professor of computer science and electrical engineering at the University of Southern California.He is also Director of the Distance Education Network in the School of Engineering. He was chairman of the Computer Science Department at USC for nine years. He is the author of 10 books and more than 100 research articles on computer science subjects ranging from data structures, algorithms, and software design to computer science education.

Dr. Raymond Madachy is an adjunct assistant professor in the computer science and industrial and systems engineering departments and research collaborator with the USC Center for Software Engineering. He is also the manager of the Software Engineering Process Group at Litton Guidance and Control Systems, which achieved SEI CMM Level 4 in December 1998. He completed his Ph.D. in Industrial and Systems Engineering at USC in 1994.

Chris Abts holds Bachelor's and Master's degrees in industrial engineering and is a research assistant/Ph.D. Candidate at the USC Center for Software Engineering. He has worked as a lead systems engineer/analyst for Logicon in the area of Mission Planning, including development of automated mission planning tools, algorithm development, mission scenario analysis, and simulation modeling. His research interests have been in software metrics and cost models, risk management, and systems architecting, including the development of the COCOTS COTS modeling extension to COCOMO II.

Information on other authors of this article may be found at http://sunset.usc.edu/Research_Group/People.html

Software Estimation: Challenges and Research

Software cost and schedule estimation supports the planning and tracking of software projects. Because software is complex and intangible, software projects have always been harder to estimate than hardware projects. In this article, we review challenges for software cost estimators, and describe research work under way to address these challenges.

The Changing Nature of Software Development

There are now many ways to develop and sell software products. This means that the scope of the estimation problem is much larger now than it was in the early days of software development. In the 1960s, all software was custom built. Projects had dedicated staff. Companies were usually paid on a level of effort basis (cost plus, or time and materials).

Today, there are many ways to build software (custom coding, integration of pre-built components, generation of code using tools, etc.). The project environment is more varied as well. Large, complex software products need a wider range of skills. Organizational structures are flatter and more fluid. Workers are often dispersed and may be hired only for short periods of time when particular skills are needed. The business environment has also become more complex. There are new ways of selling software and data, increased liabilities for defective products, and new accounting practices. The following paragraphs briefly describe these areas. Each area presents challenges to software estimators.

The technology used to build systems will continue to evolve rapidly. This technology includes hardware platforms, operating systems, data formats, transmission protocols, programming languages, methods, tools, and commercial off-the-shelf (COTS) components. The half-life of software engineering knowledge is typically less than three years.

Project teams will use new development processes such as rapid application development (RAD) and COTS integration to grow software systems. These processes will continue to evolve rapidly as we learn. (Unless an organization is at Software Engineering Institute Capability Maturity Model® Level 5, it may not be able to evaluate and inject new technology at the rate needed to keep up.) Such constant and rapid changes mean that little relevant historical data will be available to help estimate future software projects and to develop new cost estimation models. This may challenge CMM® criteria relating to estimating, planning, and tracking that require the use of historical data and assume a stable process.

Technology and processes alone do not determine how businesses will develop software, license products, etc. There is an interaction with the legal and business domains. For example, Paul Strassman discusses possible ways of acquiring software as summarized in Table 1. Strassman observes that outsourcing or renting software (data processing) capability frees an organization from the details of business support functions, and allows it to focus on business objectives and growth. For additional information, see www.strassman.com

Scott McNealy of Sun Microsystems proposes putting everyday applications on the Internet for all to use for free [1]. Such external influences will affect how software is built and sold.

The Capability Maturity Model and CMM are registered in the U.S Patent and Trademark office to Carnegie Mellon University.

Future software estimators will have to estimate the costs of trade-offs between development, operating, and maintenance costs. This will require more knowledge of financial practices such as return on investment, discounted value, etc. Barry Boehm covers such topics in Part III of his classic book [2].

Another factor that estimators must confront is the people who will produce the software. Projects need experts in multiple application and solution domains in order to build the large, complex systems. They may also have to receive guidance from experts in the legal, regulatory, and business domains. No single person can understand all of these domains, nor can one individual keep up with them all. This means that development teams will be more interdisciplinary. Because all of these experts are not needed continuously, project teams (and possibly entire companies) may consist of a core of permanent experts and groups of temporary workers hired just in time. The permanent staff would include managers, project control, chief engineers, etc. The temporary workers would include analysts, designers, engineers, testers, support staff, and various domain experts. It will be challenging to assemble and manage such diverse, dynamic teams. Advances in telecommunications, networking, and support software (groupware) can help such teams function even though they are geographically and temporally dispersed. Such organizational structures affect estimators because they impact parameters such as the average staff capability, experience, and turnover.

There will also be a growing need for estimates of quality (defects), reliability, and availability, as well as the usual cost and schedule estimates. Developers and customers will become more interested in ensuring the safety and reliability of complex software systems such as those used for financial and medical applications. Estimating such characteristics is especially challenging for systems built using COTS components.

Promising Research

Several areas hold promise for coping with these challenges. This section describes recent work.

Size

We measure size for many reasons. We use size to estimate effort, to measure memory requirements, to estimate processing or execution speed, etc. What we measure and the units we use are determined by our intended use of the measure. The goal-question-metric paradigm addresses this concept in detail [3]. The

Table 1. *Ways to Acquire Software**

- Build it yourself
- Hire a developer to build it
- Hire a firm to build, maintain, and operate it for you (outsourcing)
- Purchase COTS products
- Rent pre-built, pre-tested functions

*Adapted from a talk presented in October by Paul Strassman.

usual goal of software cost estimators is to determine the amount of effort and schedule needed to produce a software product. The amount of functionality in the product is gauged in terms of size (measured in Source Lines of Code, function points, etc.). The estimator uses some productivity model or cost estimating relation to compute effort from the size. Schedule is often computed from the total effort, possibly modified by parameters such as the rate of staff buildup, etc. (Interestingly, several models for the development of new software have schedule approximately proportional to the cube root of the total effort.)

Size in SLOC clearly depends on the choice of programming language. This leads to difficulties in defining the true amount of functionality in a software product, and in measuring programmer productivity. Capers Jones described these in his recent *Scientific American* article [4]. Jones observes that the cost of developing software is determined by more that just the cost of the actual coding process. Considerable effort is spent in designing, documenting, and testing. Jones, and others, advocates a size measure developed by Allan Albrecht called function points [5]. Albrecht's goal was to define a measure that was independent of the implementing technology and based on quantities a user could understand. He used five quantities: inputs, outputs, queries, internal logical files, and external interface files.[1] The International Function Point Users' Group (IFPUG) is the custodian of the official counting rules.[2]

Albrecht originally defined function points only for Management Information Systems. Since then, several workers have extended the concept to address other types of systems. For example, Charles Symons has extended the concept to meet the needs of transaction-based systems [6]. David Garmus discussed function point counting in a real-time environment [7].

Alain Abran and his collaborators are working to update the measures of software size by extending function points to create what they call full function points. Full function points are based on a solid theoretical foundation and will be validated using actual data from modern projects. See [8] and [9]. Full function points provide one way to measure the size of the product.

Some authors are attempting to link the products of analysis and design directly to the size measured in function points, or some variant thereof. Specifically, they endeavor to tie the attributes of diagrams of the Unified Modeling Language (UML) to function points. Some recent references are [9][3] and [10]. This will make counting of software size more objective.

> "Software is seldom built using the Waterfall Model. Still, we need some sort of milestones to be able to define the scope of project activities covered by effort and schedule estimation models."

This increased objectivity and precision comes at a price: we cannot count the size until after some analysis has been done. Such a sizing method, while more precise, could affect how software is procured. Perhaps it will be more like the way that buildings are purchased. The architect works with the user to define the building. All stakeholders must agree on the building's purpose, architectural style, types of rooms and their juxtaposition, overall size, and types of building materials. Sometimes there are additional constraints such as zoning laws, building codes, and available funding. Once everyone agrees, the architect draws up detailed plans and proceeds to cost the project (The detailed plans for software products could perhaps be UML diagrams.). The architect receives a fee computed as some percentage of the total cost of the building. Another approach is to fund the early stages of requirements analysis and product design as level-of-effort tasks. Once the team reaches product design review (PDR)[4] a binding production contract can be negotiated. This will also affect which phases and activities are covered by the estimation models.

Size is also difficult to define for prebuilt components. In many cases, the developer does not even have the source code, so measures such as SLOC are not feasible. In addition, the developer needs to understand, integrate, and test only the portion of the component's interfaces and functionality that is actually used. The size needs to reflect only this portion for the purpose of estimating the developer's effort.

Still other size measures are needed for software maintenance. Lines of code are of little use. Some possible size measures are the number of change requests, and the number of modules affected by a particular change request. Lack of time prevents us from discussing this topic further here.

Standardized Process Milestones

Software is seldom built using the Waterfall Model. Still, we need some sort of milestones to be able to define the scope of project activities covered by effort and schedule estimation models. Boehm has proposed "process anchors" as one way to standardize the description of the production process [11]. These are points where significant decisions or events happen in a project. Table 2 lists an extension of these. The milestones shown in Table 2 are adapted from the Model-Based (System) Architecting and Software Engineering (MBASE) life cycle process model [12] and [13], and the Rational Unified Process (RUP) [14], [15] and [16]. I have added the Unit Test Complete milestone.

The life cycle concept objectives (LCO) milestone defines the overall purpose of the system and the environment in which it will operate. The operational concept indicates which business or mission functions will be performed by the software and which will be performed manually by the operators. The stakeholders agree on the system's requirements and life cycle. The design team has identified a feasible architecture. The information defined at LCO is critical to guide designers and programmers as they refine, elaborate, and implement the system.

The life cycle architecture (LCA) milestone confirms the top-level structure for the product. This further constrains the choices available to the designers and implementers. The unit test complete (UTC) milestone occurs after LCA. Even for rapid development, there should come a time when a component is considered finished. (This point is also called "code complete" by some authors. This name apparently arose at Microsoft [17]. Rational Corp. has also defined an equivalent milestone.) The UTC milestone

occurs when the programmers relinquish their code because they sincerely believe that they have implemented all the required features, turning it over to the formal configuration control process. For large military projects, the UTC milestone occurs after unit testing when approximately 60-70 percent of the total effort has been expended. The initial operational capability (IOC) is the first time that the system is ready for actual use.

Inception Readiness Review (IRR)

- Candidate system objectives, scope, boundary defined
- Key stakeholders identified

Life Cycle Objectives Review (LCO)

- Life Cycle Objectives (LCO) defined (key elements of Operational Concept, Requirements, Architecture, Life Cycle Plan)
- Feasibility assured for at least one architecture
- Acceptable business case
- Key stakeholders concur on essentials

Life Cycle Architecture Review (LCA)

- Life cycle Architecture (LCA) elaborated and validated
- Product capabilities defined by increment (build content)
- Key stakeholders concur on essentials
- All major risks resolved or covered by risk management plan

Unit Test Complete (UTC)

- All identified components completed and unit tested
- All associated engineering documentation updated
- Code and engineering documentation delivered to Configuration Management (The component is complete and ready to be integrated.)

Initial Operational Capability (IOC)

- Operational and support software built, integrated, and tested.
- Appropriate documentation completed
- Operational data prepared or converted
- Necessary licenses and rights for COTS and reused software obtained
- Facilities, equipment, supplies, and COTS vendor support in place
- User, operator and maintainer training and familiarization completed

Transition Readiness Review

- Plans for full conversion, installation, training, and operational cutover complete

Product Release Review (PRR)

- Assurance of successful cutover from previous system for key operational sites

Table 2. *New Process Milestones*

Theoretical Productivity Models

We need models that tell us how product size relates to the effort (and schedule) required to build the product. For example, Shari Pfleeger describes models of software effort and productivity [18].

Unfortunately, there no longer seems to be a good measure of effort since the effort required is not a linear function of the size. Software development requires the engineer to manipulate and connect many mental models. Since information is only communicated via voice and diagrams, there is inefficiency in transferring knowledge about the system between people on the team. This means that having fewer people reduces the amount of communication effort and also the amount of distortion introduced by the loss and noise associated with the imperfect communications channel. (Standardized diagrams such as UML and formal methods attempt to improve the precision and efficiency of communication.)

Larger groups of people work more inefficiently. The main causes for this diseconomy of scale are the need to communicate complex concepts and mental models between the workers, and the need to coordinate the activities of a group of workers who are performing a set of complex, interrelated tasks. If a project has N workers who must all communicate with one another, the number of possible communication paths is N(N-1)/2. Managers create hierarchical organizations to reduce the number of direct interactions. Sometimes, however, software products are so complex that it is not easy to partition the tasks. Samuel Conte and his coworkers discuss this topic in some detail [19]. They also define the COoperating Programming MOdel (COPMO) model to explicitly compute such effects. Basically, they compute the total development effort as the sum of two terms. The first term is the effort required by individuals working independently on modules. The second term is the effort required to coordinate the activities of the team.[5] Other existing parametric models also attempt to account for the diseconomy of scale.

Parametric Models

An earlier *CROSSTALK* article described the emergence of parametric models [20]. Barry Boehm originally defined the Constructive Cost Model (COCOMO) in 1981. During the 1990s Boehm and his collaborators have worked to modernize COCOMO. See [21]. Basically, they have defined a family of estimation models to address different types of development processes. Their family presently includes:

- COCOMO–Constructive Cost Model.
- COQUALMO–Constructive QUALity Model.
- COCOTS–Constructive COTS model.
- COSSEMO–Constructive Staged Schedule and Effort Model.
- CORADMO–Constructive Rapid Application Development Model.
- COPROMO–Constructive Productivity Improvement Model.[6]

Although most of the past work defined models for cost and schedule estimation, some work on models to estimate software reliability and defect densities has also been done. As the importance of reliability increases, improved versions of such models will be needed.

Back to Basics

Planners and estimators need ways to address these challenges today. I think that there are three things we can do. First, we need to measure our projects, products, and processes. To reduce measurement costs, we need to collect and analyze data based on our goals. The goal-question-metric (GQM) model is one way to attack this. Second, we should use updated estimation models as they become available. We should also use our metrics to formulate simple estimation models that are better suited to the new project types and development processes. Because all models are only approximations, we should never rely on a single model for our estimates. Instead we should compare estimates from two or more models. Third, we must be prepared to change our metrics and models as we gain new understanding of the new development processes, technologies, and tools. This will continue to be a very dynamic and exciting area of research as we enter the new millennium.

References

1. McNealy, Scott, Stop Buying Software, *Dot Com Perspectives.* See www.sun.com/dot-com/perspectives/stop.html.
2. Boehm, Barry W., *Software Engineering Economics,* Prentice-Hall, 1981.
3. Basili, V.R. and D.M. Weiss, A Method for Collecting Valid Software Engineering Data, *IEEE Transactions on Software Engineering,* vol. 10, no. 6, pages 728-738, 1984.
4. Jones, Capers, Sizing Up Software, *Scientific American,* December 1998, pp. 104-109.
5. Albrecht, Allan J., Measuring Application Development Productivity, *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium,* October 14-17, 1979.
6. Symons, Charles, *Software Sizing and Estimating: Mark II Function Points (Function Point Analysis),* John Wiley & Sons, 1991, ISBN 0-471-92985-9.
7. Garmus, David, Function Point Counting in a Real-Time Environment, *CROSSTALK*, Vol. 9, No. 1, January 1996, pp. 11-14.
8. Abran, Alain and P.N. Robillard, Function Point Analysis: An Empirical Study of Its Measurement Processes, *IEEE Transactions on Software Engineering,* vol. 22, no. 12, December 1996, pp. 895-909.
9. IWSM'99, *Proceedings of the Ninth International Workshop on Software Measurement,* Lac Supérieur, Québec, Canada, 8-10 Sept. 8-10, 1999.
10. Stutzke, Richard D., Possible UML-Based Size Measures, *Proceedings of the 18th International Forum on COCOMO and Software Cost Modeling,* Los Angeles, Calif., Oct. 6-8, 1998.
11. Boehm, Barry W., Anchoring the Software Process, *IEEE Software,* Vol. 13, No. 4, July 1996, pages 73-82.
12. Boehm, Barry W., D. Port, A. Egyed, and M. Abi-Antoun, The MBASE Life Cycle Architecture Package: No Architecture is an Island, in P. Donohoe (ed.), *Software Architecture,* Kluwer, 1999, pp. 511-528.
13. Boehm, Barry W., and Dan Port, Escaping the Software Tar Pit: Model Clashes and How to Avoid Them, *ACM Software Engineering Notes,* January 1999, pp. 36-48.
14. Royce, Walker E., Software Project Management: A Unified Framework, Addison-Wesley, 1998.
15. Kruchten, Philippe, *The Rational Unified Process: An Introduction,* Addison-Wesley, 1999.
16. Jacobson, Ivar, Grady Booch, and James Rumbaugh, *The Unified Software Development Process,* Addison-Wesley, 1999.
17. Cusumano, Michael A. and Richard W. Selby, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, The Free Press, 1995. Page 195 describes the code complete concept.
18. Pfleeger, Shari Lawrence, Model of Software Effort and Productivity, *Information and Software Technology,* vol. 33, no. 3, April 1991, pp. 224-231.
19. Conte, Samuel D., H.E. Dunsmore, and V.Y. Shen, *Software Engineering Metrics and Models,*" Benjamin/Cummings, 1986. Section 5.8 describes task partitioning and communications overhead. Section 6.7 describes COPMO.
20. Stutzke, Richard D., *Software Estimating Technology: A Survey, CROSSTALK*, vol. 9, no. 5, June 1996, pages 17-22.
21. Boehm, Barry W, Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy and Richard Selby, Cost Models for Future Software Life Cycle Processes: COCOMO 2.0, *Annals of Software Engineering, Special Volume on Software Process and Product Measurement,* vol. 1 no. 1, pages 57-94. J. C. Baltzer AG Science Publishers, Amsterdam, The Netherlands, 1995. See http://manta.cs.vt.edu/ase/vol1Contents.html.

Notes

1. The files may actually be collections of physical files, database tables, etc. These collections are perceived externally as single entities.
2. The IFPUG web site is http://ifpug.org
3. Accessible online at: www.lrgl.uqam.ca/iwsm99/indes2.html. Three papers dealing with UML-based size measures were presented by Stutzke, Labyad et. al., and Bévo et. al.
4. I use the phrase product design review deliberately. There is nothing preliminary about it for software projects. Once PDR occurs you have defined the form into which the programmers start to pour concrete. After PDR, changes become very expensive for software!
5. There are some challenges with calibrating and using this model for prediction. See their book.
6. For more information on these models see the COCOMO II web site at http://sunset.usc.edu/COCOMOII/suite. html

About the Author

Dr. Richard D. Stutzke has more than 40 years of experience developing software. He established and led the Corporate Software Process Group for Science Applications International Corp. (SAIC) for its first two years. For the past seven years, he has been a principal member of two Software Engineering Process Groups: one at the Army Aviation and Missile Command's Software Engineering Directorate and one at SAIC's Applied Technology Group. Both organizations have achieved SEI CMM Level 3. Dr. Stutzke has written dozens of papers in the field of software cost estimation and is writing a book for Addison-Wesley, *Software Estimation: Projects, Products, Processes,* that will appear this fall.

Dr. Richard D. Stutzke
Science Applications International Corp.
6725 Odyssey Drive
Huntsville, Ala. 35806-3301
Voice: 256-971-6624 or 256-971-7330
Fax: 256-971-6550
E-mail: Richard.D.Stuzke@saic.com

Instant Notification

Would you like to be notified when the online version of *CROSSTALK* becomes available? Take a moment to link to the new e-mail update form accessible from **www.stsc.hill.af.mil** to ensure we have accurate, up-to-date information from you.

CROSSTALK online is available before the printed version, and you do not want to miss those early downloads from upcoming issues on the F-22, CMMI part I and II, Network Security, Software Acquisition, Project Managment, and many more.

Quote Marks

"I used to think that cyberspace was 50 years away. What I thought was 50 years away was only 10 years away. And what I thought was 10 years away . . . it was already here. I just wasn't aware of it yet."—BRUCE STERLING

"I think computer viruses should count as life. I think it says something that the only form of life we have created so far is purely destructive. We've created life in our own image."
— Stephen Hawking

Coming Events

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

April 15-18
ACM International Conference on Management of Data
http://www.seas.smu.edu/sigmod2000

April 18-20
FOSE
www.fedimaging.com/conferences

April 24-28
Software Engineering Australia Conference (SEA 2000)
E-mail for information: johnl@sea-act.com.au

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

May 30-June 2
Thirteenth International Software Quality and Internet Quality Week (QW 2000)
http://www.soft.com/QualWeek/QW2K/index.html

May 22-23
6th Annual Montgomery Golf Outing and Information Technology Partnership Day
http://web1.ssg.gunter.af.mil/partnership

June 4-11
22nd International Conference on Software Engineering
http://www.ul.ie/~icse2000

June 4-7
9th Biennial IEEE
http://cefc2k.aln.fiu.edu

June 5-7
2000 IEEE International Interconnect Technology Conference
http://www.his.com/~iitc

June 10-14
ISCA2000: 27th International Symposium on Computer Architecture
http://www.cs.rochester.edu/meetings/ICSA2K

June 18-22
ICC 2000—IEEE International Conference on Communications
http://www.icc00.org/

July 11-13
5th Annual Conference on Innovations and Technology in computer Science Education
http://www.cs.helsinki.fi/events/iticse

July 16-18
7th IEEE Workshop on Computers in Power Electronics
http://www.conted.vt.edu/compel.htm

July 16-19
Congress on Evolutionary Computation
http://pcgipseca.cee.hw.ac.uk/cec2000

August 6-11
6th Annual International Conference on Mobile Computing and Networking
http://www.research.telcordia.com/mobicom2000

Does Calibration Improve Predictive Accuracy?

There are many sophisticated parametric models for estimating the size, cost, and schedule of software projects; however, the predictive accuracy of these models is questionable. Several authors assert that a model's predictive accuracy can be improved by calibrating (adjusting) its default parameters to a specific environment. This article reports the results of a three-year, 10-study project that tests this assertion. Results show that calibration did not improve the predictive accuracy of most of the models we tested. In general, the accuracy was no better than within 25 percent of actual development cost or schedule, about one half of the time.

Software costs continue to rise in the Department of Defense (DoD) and other government agencies. To better understand and control these costs, agencies often use parametric cost models for software development cost and schedule estimation. However, the accuracy of these models is poor when the default values embedded in the models are used [1]. Even after the software cost models are calibrated to DoD databases, most have been shown to be accurate to within only 25 percent of actual cost or schedule about half the time. For example, Robert Thibodeau [2] reported accuracy of early versions of the PRICE-S and SLIM models to be within 25 and 30 percent, respectively, on military ground programs. The IIT Research Institute [3] reported similar results on eight Ada programs, with the most accurate model at only 30 percent of actual cost or schedule, 62 percent of the time.

Furthermore, the level of accuracy reported by these studies is likely overstated because most studies have failed to use holdout samples to validate the calibrated models. Instead of reserving a sample of the database for validation, the same data used to calibrate the models were used to assess accuracy [4]. In a study using 28 military ground software data points, Gerald Ourada [5] showed that failure to use a holdout sample overstates a model's accuracy. Half of the data was used to calibrate the Air Force's REVIC model. The remaining half was used to validate the calibrated model. REVIC was accurate to within 30 percent, 57 percent of the time on the calibration subset, but only 28 percent of the time on the validation subset.

Validating on a holdout sample is clearly more relevant because new programs being estimated are, by definition, not in the calibration database. The purpose of this project was to calibrate and properly evaluate the accuracy of selected software cost estimation models using holdout samples. The expectation is that calibration improves the estimating accuracy of a model [6].

The Decalogue Project

This paper describes the results of a long-term project at the Air Force Institute of Technology to calibrate and validate selected software cost estimation models. Two Air Force product centers provided software databases: the Space and Missile Systems Center (SMC), and the Electronic Systems Center (ESC). The project has been nicknamed the "Decalogue project" because 10 masters' theses extensively document the procedures and results of calibrating each software cost estimation model.

The Decalogue project is organized into three phases, corresponding to when the theses were completed. Five theses were completed in 1995; two theses were completed in 1996; and three theses were completed in 1997. Lessons learned during each phase were applied to the next phase. A brief description of each phase and its results follows.

Phase 1

Each student calibrated a specific software cost model using the SMC software database. The models were the Revised Enhanced Intermediate Version of the Constructive Cost Model (COCOMO) (REVIC), the Software Architecture Sizing and Estimating Tool (SASET), PRICE-S, SEER-SEM, and SLIM. The government owns REVIC and SASET. The other models are privately owned.

Management Consulting and Research developed the SMC database, and it contains detailed historical data for more than 2,500 software programs. The database includes inputs for REVIC, SASET, PRICE-S, and SEER-SEM for some of the 2,500 projects, but none specifically for SLIM.

The details of each thesis project are described in the separate thesis reports [7, 8, 9, 10, 11]. Each is available from the Defense Technical Information Center. Additional detail is also available from the authors of this article [12, 13]. Here, only the highlights of the results of the five studies are provided.

Calibration rules. The five models were calibrated to a portion of the SMC database. The database was divided into subsets: military ground, avionics, unmanned space, missiles, and military mobile. The military ground subset was further divided into command and control programs and signal processing programs. Each subset was divided into calibration and holdout samples using three rules:

1. If there were less than nine data points, the subset was considered too small for a holdout sample and could not be validated.
2. If there were between nine and 11 data points, eight were randomly selected for calibration and the rest were used for validation.
3. If there were 12 or more data points, two-thirds were randomly selected for calibration and the rest were used for validation.

The accuracy of each model was evaluated using criteria proposed by Samuel Conte, et al. [14] based on the following statistics:

(1) Magnitude of Relative Error (MRE) = | Estimate – Actual | / Actual
(2) Mean Magnitude of Relative Error (MMRE) = (MRE) / n
(3) Root Mean Square (RMS) = $[(1/n)(\text{Estimate} - \text{Actual})^2]^{1/2}$
(4) Relative Root Mean Square (RRMS) = RMS / [(Actual) / n]
(5) Prediction Level (Pred (.25)) = k/n

For Equation No. 5, n is the number of data points in the subset and k is the number of data points with MRE # 0.25. According to Conte, et al. [14], a model's estimate is accurate when MMRE < 0.25, RRMS < 0.25, and Pred (.25) > .75.

Results. Table 1 summarizes the results of Phase 1. Due to an oversight,

not all five theses reported RRMS. Thus, only MMRE and PRED (.25) are shown. Validation sample size is the number of data points in the holdout sample used for validation. For some models, the military ground subsets (signal processing and command and control) were combined into an overall military ground subset to obtain a sufficiently large sample size for validation.

As shown in Table 1, most of the calibrated models were inaccurate. In the two instances where the calibrated models met Conte's criteria, only one data point was used for validation. Thus, these results are not compelling evidence that calibration improves accuracy. In some cases the calibrated model was less accurate than the model before calibration.

These results may be due in part to the nature of the databases available to DoD agencies. In the SMC database, the developing contractors are not identified. Therefore, the data may represent an amalgamation of many different development processes, programming styles, etc., which are consistent within contracting organizations, but vary widely across contractors. Also, because of inconsistencies in software data collection among different DoD efforts, actual cost data and other data may be inconsistent and unreliable.[1]

Phase 2

In 1996 two additional models, SoftCost-OO and CHECKPOINT, were calibrated by two master's students. CHECKPOINT is unique among the models calibrated in this study because the internal algorithms are based on function points instead of lines of code.[2] Details are provided in their thesis reports [15,16]. A brief description of each model, the calibration procedures, and the results of Phase 2 follow.

Calibration rules. With a few exceptions related to the subsets to calibrate and the holdout sample rules, the two models were calibrated and validated using the same methods that were used in Phase 1. A seventh subset of the SMC database, ground in-support-of-space ("Ground Support" in Tables 2 and 3) was used for both models. For SoftCost-OO, three additional subsets for European Space Agency programs were added, since SoftCost-OO is used extensively in Europe.

Table 1 **REVIC, SASET, PRICE-S, SEER-SEM, AND SLIM CALIBRATION RESULTS (1995)**

Model	Data Set	Validation Sample Size	Pre-Calibration MMRE	Pre-Calibration PRED (.25)	Post-Calibration MMRE	Post-Calibration PRED (.25)
REVIC	Military Ground	5	1.21	0	0.86	0
	Unmanned Space	4	0.43	0.50	0.31	0.50
SASET	Avionics	1	1.76	0	0.22*	1.00*
	Military Ground	24	10.04	0	0.58	0
PRICE-S	Military Ground	11	0.30	0.36	0.29	0.36
	Unmanned Space	4	0.34	0.50	0.34	0.50
SEER-SEM	Avionics	1	0.46	0	0.24*	1.00*
	Command and Control	7	0.31	0.43	0.31	0.29
	Signal Processing	7	1.54	0.29	2.10	0.43
	Military Mobile	4	0.39	0.25	0.46	0.25
SLIM	Command and Control	3	0.62	0	0.67	0

* Met Conte's criteria

Table 2 **SOFTCOST CALIBRATION RESULTS (1996)**

Data Set	Validation Sample Size	Pre-Calibration MMRE	Pre-Calibration RRMS	Pre-Calibration PRED (.25)	Post-Calibration MMRE	Post-Calibration RRMS	Post-Calibration PRED (.25)
Ground Support	15	2.73	3.13	0.13	1.80	1.96	0.20
Ground Support (Europe)	25	3.05	3.61	0.08	0.67	0.84	0.36
Unmanned Space	5	0.56	1.05	0.20	0.48	0.92	0.20
Unmanned Space (Europe)	7	1.79	0.79	0.14	1.27	0.84	0.14
Avionics	5	0.71	0.76	0.20	0.85	0.56	0.20
Command and Control	6	1.90	3.43	0.17	0.52	0.87	0.50
Signal Processing	9	0.43	0.61	0.11	0.28	0.64	0.44
Military Mobile	5	0.63	0.51	0.20	0.42	0.40	0.20

Table 3 **CHECKPOINT CALIBRATION RESULTS (EFFORT, 1996)**

Data Set	Validation Sample Size	Pre-Calibration MMRE	Pre-Calibration RRMS	Pre-Calibration PRED (.25)	Post-Calibration MMRE	Post-Calibration RRMS	Post-Calibration PRED (.25)
Effort – Function Points							
MIS – COBOL	6	0.54	0.10	0.67	0.02*	0.01*	1.00*
Military Mobile - Ada	4	1.38	0.41	0.25	0.19*	0.06*	0.75*
Avionics	4	0.82	0.68	0.50	0.16*	0.11*	0.75*
Effort – SLOC							
Command and Control	6	0.19*	0.14*	0.50	0.16*	0.16*	0.50
Signal Processing	10	0.09*	0.08*	1.00*	0.09*	0.08*	1.00*
Unmanned Space	5	0.05*	0.05*	1.00*	0.04*	0.06*	1.00*
Ground Support	4	0.05*	0.06*	1.00*	0.05*	0.06*	1.00*
COBOL Programs	4	0.05*	0.05*	1.00*	0.05*	0.05*	1.00*

* Met Conte's Criteria

Table 4 **CHECKPOINT CALIBRATION RESULTS (1997)**

Data Set	Validation Sample Size	Pre-Calibration MMRE	Pre-Calibration RRMS	Pre-Calibration PRED (.25)	Post-Calibration MMRE	Post-Calibration RRMS	Post-Calibration PRED (.25)
Ada Language	8	1.21	1.34	0.00	1.70	2.54	0.50
Assembly Language	11	0.83	1.44	0.09	2.05	1.20	0.18
FORTRAN Language	12	0.73	1.12	0.17	0.70	2.31	0.17
JOVIAL Language	7	0.71	1.22	0.00	0.44	0.68	0.43
Contractor B	4**	0.60	0.74	0.13	0.64	0.49	0.25
Contractor J	11	0.69	0.91	0.18	1.33	1.43	0.18
Ada and Contractor R	5**	0.59	0.57	0.05	0.39	0.72	0.45
CMS2 and Contractor M	5**	0.91	1.13	0.00	0.69	0.64	0.10
FORTRAN and Contractor A	7	0.82	0.84	0.00	0.44	0.88	0.29
JOVIAL and Contractor J	6	0.80	1.42	0.00	0.37	0.70	0.33

** Resampling Used for This Set

Table 5 **COCOMO II CALIBRATION RESULTS (1997)**

Data Set	Total Sample Size	Pre-Calibration MMRE	Pre-Calibration RRMS	Pre-Calibration PRED (.25)	Post-Calibration MMRE	Post-Calibration RRMS	Post-Calibration PRED (.25)
Command and Control	12	0.39	0.49	0.30	0.33	0.53	0.40
Signal Processing	19	0.45	0.63	0.33	0.38	0.53	0.40
Ground Support	15	0.71	1.16	0.07	0.66	0.95	0.20
Military Mobile	12	0.79	0.95	0.10	0.68	0.74	0.00

Table 6 **SAGE CALIBRATION RESULTS (1997)**

Data Set	Total Sample Size	Pre-Calibration MMRE	Pre-Calibration RRMS	Pre-Calibration PRED (.25)	Post-Calibration MMRE	Post-Calibration RRMS	Post-Calibration PRED (.25)
SMC – Avionics	9	0.45	0.54	0.21	0.39	0.52	0.24
Command and Control	10	0.23*	0.23*	0.70	0.29	0.30	0.45
Signal Processing	16	0.39	0.43	0.44	0.50	0.54	0.20
Unmanned Space	7	0.66	0.69	0.14	0.59	0.88	0.30
Ground Support	14	0.32	0.44	0.43	0.32	0.44	0.43
Military Mobile	10	0.37	0.47	0.29	0.41	0.52	0.36
Missile	4	0.66	0.89	0.00	0.67	0.44	0.24
ESC – Contractor A	17	0.48	0.57	0.17	0.41	0.40	0.31
Contractor J	17	0.37	0.47	0.33	0.47	0.57	0.14
Contractor R	6	0.32	0.36	0.32	0.21*	0.23*	0.54

* Met Conte's Criteria

For CHECKPOINT, the missile subset was not used, and no European programs were used. In addition, data were obtained on Management Information System programs written in Common Business-Oriented Language (COBOL) from a local contractor, and a subset for COBOL programs was added to determine if stratification by language would provide better results. Finally, the rules to determine the sizes of the calibration and holdout samples were changed to avoid the problem of single-point validations experienced in Phase 1. If there were eight or more data points in a subset, half were used for calibration, and the other half for validation. If there were fewer than eight data points, that subset was not used.

Results. Table 2 and Table 3 show the results of calibrating each model for development effort. For SoftCost-00 (Table 2), calibration almost always improved the accuracy of the model, although none of the subsets met Conte's criteria. For CHECKPOINT, all but one subset met the criteria when predicting development effort (Table 3).

Since CHECKPOINT uses function points as a measure of size, they were used when sufficient data points were available for the subsets; otherwise, source lines of code (SLOC) were used. For three function point effort subsets, there was substantial improvement in accuracy after the model was calibrated for other programs in these subsets, especially for the Management Information System COBOL subset. Except for the Command and Control subset, the SLOC effort subsets met Conte's criteria before and after calibration. Although calibration did not significantly improve accuracy for these subsets (primarily because SLOC are an output, not an input, to CHECKPOINT), the accuracy was good even without calibration. CHECKPOINT results for effort estimation are especially noteworthy as inputs for this model were not considered when the SMC database was developed.

Although these results were promising, it should not be assumed that CHECKPOINT will do as well in other environments. The best results for the CHECKPOINT model were for the Management Information System COBOL data set, which was obtained from a single contractor. Data from multiple contractors, which often characterize DoD databases, are more difficult to calibrate accurately. Furthermore, CHECKPOINT is a function point model. If the user wants to input size in SLOC, which is usually the case, the user or model must first convert the SLOC to function points. Unfortunately, the conversion ratios are sometimes subject to significant variations. Thus, the SLOC effort results for CHECKPOINT may not work out as well elsewhere.

Phase 3

In 1997 three models (COCOMO II, SAGE, and CHECKPOINT) were calibrated. COCOMO II, the successor to Boehm's COCOMO model [1], was calibrated to the SMC database. The SAGE model, a commercial model developed by Randy Jensen, was calibrated to the SMC and ESC databases. Finally, CHECKPOINT was calibrated to the ESC database to determine whether the unusually high accuracy reported by Karen Mertes [15] could be achieved on a different database. As before, the details are documented in the 1997 thesis reports [17,18,19]. Only the highlights are described here.

The ESC database contains information on 52 projects and 312 computer software configuration items [18], and contractor identifiers and language, but does not contain information on application type. It also contains inputs for the SEER-SEM model for which it was originally developed. The ESC database was initially stratified by a contractor as it was thought that a model could be more accurate when calibrated for a specific developer [6]. For CHECKPOINT, the ESC database was also stratified by language, and by contractor and language.

Calibration rules. The techniques used to calibrate the models were significantly improved over those used in the earlier phases. In the past, small data sets reduced the meaningfulness of the calibration. Making statistically valid inferences from small data sets of completed software projects is a common limitation of any calibration study. To overcome this limitation, each model was calibrated multiple times by drawing random samples from the data set. The remaining holdout samples were used for validation. Averages of the validation results became the measure of accuracy. This "resampling" technique is becoming an increasingly popular and acceptable substitute for more conventional statistical techniques [20].

The resampling technique is flexible. For CHECKPOINT, resampling was used on only the small data sets (eight to 12 data points). Four random samples from the small sets were used to calibrate and validate the model. For COCOMO II, only data sets of 12 or more data points were used, and resampling was accomplished on all sets by using 80 percent of the randomly selected points for calibration, and the remaining 20 percent for validation. The process was repeated five times, and the results were averaged. For the SAGE model, all data sets having four or more points were used with an even more comprehensive resampling procedure. Simulation software, Crystal Ball, was used to select two data points for validation, and the rest for calibration. Instead of limiting the number of runs to four or five, all possible subsets were run.

Results. Table 4 shows the results of the CHECKPOINT calibration using the ESC database. Unlike the results reported by Mertes [15], none of the data sets met any of Conte's criteria, even those for a single contractor. This may be due in part to the lack of function point counts in the ESC database; only SLOC are provided for all data points. However, since Mertes' results using CHECKPOINT for SLOC were also good, it is difficult to account for differences between the results of Mertes [15] and Thomas Shrum [19].

Table 5 shows the results for COCOMO II, where calibration slightly improved the model's predictive accuracy, but none of the subsets met Conte's criteria. It is possible that better results may be attained when the online calibration capability is incorporated into the model.

Table 6 shows the results for SAGE on both databases. Although calibration sometimes resulted in improved accuracy, only a few sets met Conte's criteria. This is somewhat surprising for the ESC data sets, where individual contractors are identified by a code letter, and the model should be consistent for a company. It may be that even within a single company software

programs are developed differently. Also, it is possible that if the simultaneous effort and schedule calibration capability that is now integrated into SAGE was used, the results would be better.

Conclusion

Calibration does not always improve a model's predictive accuracy. Most of the calibrated models evaluated in this project failed to meet Conte's criteria. According to Mertes, the one exception was the calibration of CHECKPOINT to the SMC database, where almost all of the calibrated data sets met Conte's criteria for function point and SLOC applications. Unfortunately, Shrum could not replicate this result on the ESC database using a superior validation technique. Overall, none of the models was shown to be more accurate than within 25 percent of actual cost or effort, one half of the time.

This does not mean the Decalogue project was done in vain. Much was learned about the models, their strengths and weaknesses, and the challenges in calibrating them to DoD databases. One major insight is that the use of a holdout sample is essential for meaningful model calibration. Without a holdout sample, the predictive accuracy of the model is probably overstated. Since all new projects are outside of the historical database(s), validation is much more meaningful than the more common practice of analyzing within-database performance. The calibrations performed in 1997 also developed and applied resampling as a superior technique to use in validating small samples. It is better than just using one subset of data for a holdout, and can be done easily with modern software, such as Excel and Crystal Ball. Hopefully, the findings of the Decalogue project will inspire additional effort in the area of model calibration, and more promising results will be obtained.

References

1. Boehm, Barry W. *Software Engineering Economics.* Englewood Cliffs, N.J., Prentice-Hall (1981).
2. Thibodeau, Robert. *An Evaluation of Software Cost Estimating Models.* Huntsville, Ala., General Research Corp. (1981).
3. IITRI. *Test Case Study: Estimating the Cost of Ada Software Development.* Lanham, Md., IIT Research Institute (1989).
4. Ourada, Gerald L. and Daniel V. Ferens. Software Cost Estimating Models: A Calibration, Evaluation, and Comparison. *Cost Estimating and Analysis: Balancing Technology and Declining Budgets.* N.Y., Springer-Verlag, pp. 83-101 (1992).
5 Ourada, Gerald L. Software Cost Estimating Models: A Calibration, Evaluation, and Comparison. Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1991).
6. Kemerer, Chris F. An Empirical Validation of Software Cost Estimation Models. *Communications of the ACM*, pp. 416-429 (May 1997).
7. Galonsky, James C. *Calibration of the PRICE-S Model.* Master's thesis. Dayton, Ohio: Air Force Institute of Technology (1995).
8. Kressin, Robert K. *Calibration of the Software Life Cycle Model (SLIM) to the Space and Missile Systems Center Software Database (SWDB).* Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1995).
9. Rathmann, Kolin D. *Calibration of the System Evaluation and Estimation of Resources Software Estimation Model (SEER-SEM) for the Space and Missile Systems Center.* Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1995).
10. Vegas, Carl D. *Calibration of the Software Architecture Sizing and Estimation Tool (SASET).* Master's thesis. Dayton, Ohio,: Air Force Institute of Technology (1995).
11. Weber, Betty G. *A Calibration of the REVIC Software Cost Estimating Model.* Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1995).
12. Ferens, Daniel V., and David S. Christensen. Software Cost Model Calibration, An Air Force Case Study. *Journal of Cost Analysis and Management,* pp. 43-46 (Fall 1997).
13. Ferens, Daniel V., and David S. Christensen. Calibrating Software Cost Models to DOD Databases—A Review of 10 Studies. *Journal of Parametrics* 14: 33-52 (Spring 1999).
14. Conte, S.D., Dunsmore, H.E., and Shen, V.Y. *Software Engineering Metrics and Models.* Menlo Park, Calif.: Benjamin-Cummings (1986).
15. Mertes, Karen R. *Calibration of the CHECKPOINT Model to the Space and Missile Systems Center (SMC) Software Database (SWDB).* Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1996).
16. Southwell, Steven V. *Calibration of the SoftCost-R Software Cost Model to the Space and Missile Systems Center (SMC) Software Database (SWDB).* Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1996).
17. Bernheisel, Wayne A. *Calibration and Validation of the COCOMO II Cost/Schedule Estimating Model to the Space and Missile Systems Center Database.* Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1977)
18. Marzo, David B. *Calibration and Validation of the SAGE Cost/Schedule Estimating System to United States Air Force Databases.* Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1997).
19. Shrum, Thomas C. *Calibration and Validation of the CHECKPOINT Model to the Air Force Electronic Systems Center Software Database.* Master's thesis. Dayton, Ohio, Air Force Institute of Technology (1997).
20. University of Maryland. *The Resampling Project.* College Park, Md. (1997).
21. Albrecht, A.J. and J.E. Gaffney. Software Function, Source Lines of Code, and Development Effort Production: A Software Science Validation. *IEEE Transactions on Software Engineering.* Volume SE-9 (November 1983).

Notes

1. This problem was addressed in Phase 3 of the Decalogue project, where the ESC database was used. The ESC database contains an identifier for each contributing contractor.
2. Function points are weighted sums of five attributes or functions of a software program (inputs, outputs, inquiries, interfaces, and master files). Based on their analysis of more than 30 data processing programs, Allan J. Albrecht and John Gaffney[21] report that function points may be superior to SLOC as predictors of software development cost or effort.

See p. 24 for author information.

Congratulations to *CrossTalk* Top 10 Authors of 1999

CrossTalk would like to single out its Top 10 authors of the past year for their contribution to the Department of Defense's premier software engineering journal. The authors, and the name of those articles that received the highest Web hits, can be accessed at http://www.stsc.hill.af.mil *by clicking on past issues of CrossTalk.*

10 *Software Mini-Assessments: Process and Practice* by Gary Natwick, Geoff Draper, and Lennis Bearden [October 1999].

Gary Natwick is the metrics leader for the Engineering Process Group (EPG) responsible for advancing the Harris Information Systems Division to SEI SW-CMM Level 4. Previously, he was the leader of the SEPG advancing the HISD software process maturity to SEI SW-CMM Level 3. He has more than 25 years of software engineering experience (management, development, and process improvement) with Harris Corp. and the Air Force. He earned a bachelor of science degree in electrical engineering from the University of Miami. Natwick is a member of the Institute of Electrical and Electronics Engineers and the Association for Computing Machinery and is an Authorized Lead Assessor in the SEI Cost Benefits Analysis-Internal Process Improvement method.
E-mail: gnatwick@harris.com

Geoff Draper is the software focus leader of the EPG responsible for HISD software process definition and improvement. He has more than 15 years experience with Harris Corp. in various software development and leadership positions. Draper earned a bachelor's degree and a master of science degree in computer science from the University of Illinois and Florida Institute of Technology, respectively.
E-mail: gdraper@harris.com

Lennis Bearden was the leader of the EPG responsible for all HISD engineering process improvements. He has more than 25 years experience covering all aspects of system development, including hardware, software, system engineering, and program management. His interests are software processes, systems engineering process, and systems architecture. Bearden earned a bachelor's degree and master of science degree in electrical engineering from the University of Tennessee.

9 *Structured Approaches to Managing Change,* by **Mark C. Paulk** [November 1999]. Paulk is a senior member of the SEI technical staff. He has been with the SEI since 1987 and initially worked on the Software Capability Evaluation project. He has worked with the Capability Maturity Model® project since its inception and was the project leader during the development of v. 1.1 of the CMM®. He is a contributor to the International Organization for Standardization's Software Process Improvement and Capability Determination (SPICE) project, which is developing a suite of international standards for software process assessment. Prior to his work with the SEI, Paulk was a senior systems analyst for System Development Corp. (later Unisys Defense Systems) at the Ballistic Missile Defense Advanced Research Center in Huntsville, Ala. He is a senior member of the Institute of Electrical and Electronic Engineers and a senior member of the American Society for Quality Control. Paulk has a master's degree in computer science from Vanderbilt University and a bachelor's degree in mathematics and computer science from the University of Alabama in Huntsville. E-mail: mcp@sei.cmu.edu

8 *Improving Software Engineering Practice"* by **Patricia Sanders** [January 1999]. Sanders is the director of test, systems engineering, and evaluation for the Department of Defense (DoD), where she is responsible for ensuring effective integration of all engineering disciplines into the system acquisition process, including design, production, manufacturing and quality, acquisition logistics, modeling and simulation, and software engineering, with emphasis on test and evaluation as the feedback loop. She also oversees the DoD's Major Range and Test Facility Base and development of test resources such as instrumentation, targets, and other threat simulators. She has more than 24 years experience. She holds a doctorate in mathematics from Wayne State University and is a graduate of the Senior Executive Fellow Program, John F. Kennedy School of Government, Harvard University. E-mail: zetterbt@acq.osd.mil

7 *16 Critical Software Practices for Performance-Based Management,* by **Jane T. Lochner** [October 1999]. Lochner is a 1984 Naval Academy graduate. She served aboard USS Norton Sound (AVM-1) and USS Cape Cod (AD-43). She was selected to the Engineering Duty community in 1988. She has extensive experience with developing and fielding complex, real-time combat systems on aircraft carriers and large-deck amphibious ships. She is assigned to the Office of the Assistant Secretary of the Navy for Research, Development, and Acquisition working command, control, communications, computers, intelligence, surveillance, and reconnaissance and interoperability issues. She holds a bachelor's degree in marine engineering, a master's degrees in logistics, applied physics, and computer science, and is a graduate of the Defense Systems Management College Program Manager's course. E-mail: lochner.jane@hq.navy.mil

6 *Influential Men and Women of Software,* by **Kathy Gurchiek** [December 1999]. Gurchiek is the Managing Editor of *CrossTalk:* The Journal of Defense Software Engineering. She has been a journalist for nearly 20 years. She has worked as a reporter and editor for newspapers in Indiana, Illinois, and Georgia. She has served as an adjunct college instructor of communications in Salt Lake City, and her free-lance writing experience includes working for regional magazines, The Chicago Tribune, the Salt Lake bureau of The Associated Press, the Salt Lake Tribune, and the former CompuServe Wow! online magazine. She holds a master's degree in journalism from Columbia College in Chicago.
E-mail: kathy.gurchiek@hill.af.mil

5 *High-Leverage Best Practices—What Hot Companies are Doing to Stay Ahead and How DoD Programs can Benefit,* by **Norm Brown** [October 1999]. Brown is the founder and executive director of the Software Program Managers Network, a member (ex officio) of the DoD Software Management Review Board, and executive director of the DoD Software Acquisition Best Practice Initiative. Brown has a bachelor's degree in electrical engineering from Pratt Institute, a master's degree in electrical engineering from Rutgers University, and a doctor of laws from American University. He is a member of IEEE and ACM.
E-mail: spmn@aol.com

4 *Earned Value Project Management: An Introduction* by **Quentin W. Fleming** and **Joel M. Koppelman** [July 1999]. Fleming is a senior staff consultant to Primavera Systems Inc. and has more than 30 years industrial project management experience. He held various management assignments with the Northrop Corp. from 1968-91, served on an earned value corporate review team, and wrote the corporate policy directive on scheduling. He was president of the Orange County Project Management Institute (PMI) chapter and developed and taught four PMI Project Management Professional tutorial courses covering scope, cost, time, and procurement management. He has a bachelor's and a master's degree in management and is the author of seven published textbooks, including *Earned Value Project Management,* with Koppelman. E-mail:QuentinF@Primavera.com

Joel M. Koppelman is president of Primavera Systems Inc., which provides a family of project management software products. Before co-founding Primavera in 1983, he spent more than 12 years planning, designing, and managing major capital projects in the transportation industry, including duties as vice president and chief financial officer for Transportation and Distribution Associates Inc. Prior to that, he was affiliated with the management consulting firm of Booz Allen Hamilton Inc. Koppelman is a registered professional engineer with a bachelor's degree in civil engineering from Drexel University and a master's degree in business administration from the Wharton School of the University of Pennsylvania. He is a frequent speaker at universities and for international management organizations. E-mail: JKoppel@Primavera.com

3 *Confusing Process and Product: Why the Quality is not There Yet,* by David Cook [July 1999]. Cook is a principal member of the technical staff, Charles Stark Draper Laboratory, and working under contract to the Software Technology Support Center. He has more than 25 years experience in software development and has lectured and published articles on software engineering, requirements engineering, Ada, and simulation. He has been an associate professor of computer science at the Air Force Academy, deputy department head of software engineering at the Air Force Institute of Technology, and chairman of the Ada Software Engineering Education and Training Team. He has a doctorate in computer science from Texas A&M University and is a SEI-authorized PSP instructor. E-mail: cookd@software.hill.af.mil

2 *CCB—An Acronym for Chocolate Chip Brownies?* by **Reed Sorensen** [March 1999]. Sorensen is on the technical staff at TRW and was under contract to the STSC at the time this article appeared. He has more than 20 years experience developing and maintaining software and documentation of embedded and management information systems, providing systems engineering and technical assistance to program offices, and consulting with many DoD organizations regarding their software configuration management and documentation needs.

E-mail: Reed.Sorensen@cti-net.com

1 *Configuration Management: Coming of Age in the Year 2000* by **Clive Burrows** [March 1999]. Burrows is principal evaluator of configuration management products for Ovum in London, England. He is the author of four Ovum reports on this subject, the most recent was *Ovum Evaluates: Configuration Management* published June 1998.

E-mail: clive_burrows@compuserve.com

CROSSTALK will honor its Top 10 Authors of 1999 during the Software Technology Conference in Salt Lake City, Utah. Please join us at STC 2000, scheduled for April 30-May 4, when these authors are singled out for their contribution to CROSSTALK: The Journal of Defense Software Engineering. For more information on the conference, please see http://www.stc-online.org

Reducing Bias in Software Project Estimates

"It's tough to make predictions, especially about the future." – **Yogi Berra**

Nearly every software development estimate has been, or will be, biased. Biases in the estimating process contribute to poor estimates, which can affect the success or failure of a project. To understand the psychological impact of bias in developing software project task level effort estimates is essential for information technology Project Managers and their teams. The key questions become:

1. How do biases affect bottom-up task level effort estimates for software development?
2. What bias-reduction strategies can you employ to improve the quality of your estimates?

In spite of impressive advances in processes and tools, software project estimating remains more of an art than a science. Software projects continue to finish behind schedule and over budget, if they finish at all. According to a recent study, only 37 percent of software projects are completed on time and only 42 percent are completed within budget [1]. This is due in part to the difficulties in acquiring accurate estimates of software development effort.

"The subject of software estimating is definitely a black art," says Lew Ireland, former president of the Project Management Institute. Furthermore, understanding the role of judgment and bias in software estimating is even more elusive. An extensive literature search yielded virtually no research and few articles dealing with the topic. Therefore, we applied Amos Tversky and Daniel Kannamen's seminal research done in the areas of judgment and bias to the topic of software project estimating [2].

The Judgement-Bias Curve

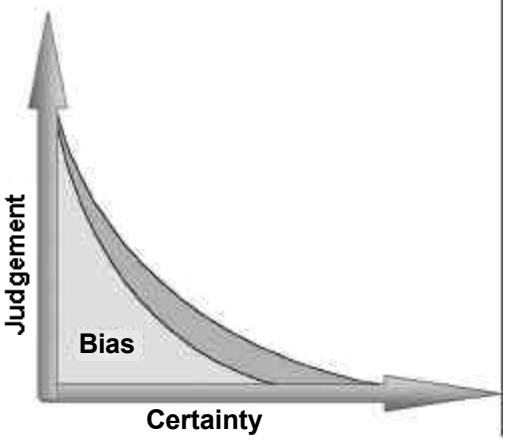
One of the most widely used methods of estimating software development tasks from a bottom-up or task-level approach is expert judgment, sometimes known as a "best guess" [3]. Expert judgment, although a very valuable method, is subject to human biases. Biases are more pronounced in the development of bottom-up estimates because expert judgment, by its very nature, is a very subjective estimating method. The estimates are most often developed through the use of best guesses due to the relative immaturity of software development as an engineering discipline. In many cases, the expert judgment estimate is produced by team members experienced in the work at hand, but not necessarily experienced in estimating techniques.

When estimating, our judgment has a fairly large degree of uncertainty associated with it. By incorporating the bias-reduction techniques outlined in this paper you can increase the level of certainty inherent within your estimates, move down the judgment-bias curve, and ultimately improve the quality of your estimates (See Figure 1.). Software estimating is more of an art than a science, and inherently more prone to the negative aspects of human biases.

Description of Bias, Example, and Bias-Reduction Strategies

The mental shortcuts, or heuristics, we use to solve complicated, uncertain problems, like estimating software development work effort, are subject to biases. A bias is a partiality, or prejudice, underlying the decision-making process (the bias emanates from the heuristic) [4]. Some biases that have an impact on the development of task-level software project estimates, particularly when derived from expert judgment, are:

Figure 1. *The Judgement-Bias Curve*



- Availability bias.
- Representative bias.
- Overconfidence bias.
- Confirmation bias.
- Insufficient anchor-adjustment bias.
- Prudence bias.

Availability Bias

This reflects our unconscious attempts to predict the future based on memories of the past. The fact that our memory is marked by more vivid or recent experiences allows the availability bias to skew our judgment. A common example is the tendency for individuals to overestimate the occurrence of a more memorable or graphic cause of death, such as a plane crash, rather than a less memorable event, such as a car crash.

Estimates are also subject to our own bounded rationality [5]. The first reasonable number that seems to make sense is often used as the starting point for an initial estimate, which often acts as an anchor (See insufficient anchor adjustment bias). The search for information on which to base this estimate is less than rigorous and often subject to mental shortcuts.

Software estimates based on expert judgment are often derived from estimating by analogy, either formally through the use of historical data or informally from past experience. Many experienced software engineers use completed projects, particularly projects they have worked on in the past, as a heuristic for current software development estimates. The availability bias predicts that information recalled from memory that is used to develop task-level estimates is most likely the very best or the very worst memories of completed tasks or projects, since these experiences would be most readily remembered. Vivid, compelling, or otherwise interesting instances from past projects can bias the estimate for the project or task.

Availability Example

Consider the case of Alex. He works on your project team, but spends most of his time talking about the Kennedy assassination and the Challenger disaster. He also refers to the last project he worked on

as the "best darn project ever done on time and under budget." Alex uses his experience on his last project as an analogy to come up with his task-level estimates for your project, which is good news. The bad news is that the estimate may be biased due to the fact that a cross-section of projects was not used.

Bias-Reduction Strategies

There are tactics you can use to stress-test Alex's estimates:

1. Ask him what assumptions were used, and whether they make sense. Basing the estimate on a similar task completed for a past project where everything (or nothing) went well is a recipe for disaster. Not only should the task be similar, but the project should be, too.
2. Encourage Alex to adjust his initial estimate so it is based on historical data or metrics such as productivity rates and size measures, if available. The objective is to use more than one project as a reference.
3. Discuss Alex's estimate as a team. Groups have been shown to exhibit less of an availability bias than individuals [6].

Representative Bias

The representative bias is most often expressed in software estimating as insensitivity to base-rate data. We can think of base-rate data as existing metric data from past projects. Individuals may tend to ignore metric data in assessing their estimates when other anecdotal information is available, even if the anecdotal information is irrelevant [7]. The representative bias predicts that even if we have extensive metric data, our teammates may not be inclined to consider it when coming up with their estimates. Under this bias, the estimate is probably constructed using information with a higher degree of uncertainty than would have been the case had we used the existing metric data.

Representative Example

Ralph has just joined your project team from another organization that did not have historical data or metrics. He avoids metrics like the plague and points out that "past experience is a poor indicator of future results." To no one's surprise, Ralph does not use historical project data or other metrics to derive his task-level estimates for your project.

Bias-Reduction Strategies

As the project manager, you are ultimately responsible for the accuracy of Ralph's estimates. To ensure you will not be in a soup line any time soon:

1. Encourage Ralph to use any and all historical data and metric information available. While intuition is good, so are data.
2. At the very least adjust Ralph's estimate in the direction of the historical average. We are more likely to perform closer to the average on subsequent trials. Statisticians call this "regression to the mean."
3. Make sure Ralph does his homework before presenting this estimate to the team. Groups generally show a higher rate of representative bias than individuals. In other words, groups are less likely to use available data and metrics [6].

Overconfidence Bias

This bias (sometimes referred to as the optimistic bias) demonstrates that we tend to overestimate our abilities and underestimate the effort involved in completing a difficult task, particularly when we are completing the task ourselves. Studies have shown that the more difficult and uncertain a task, the more prevalent the overconfidence bias [7]. In other words, individuals tend to drastically underestimate large, complex tasks when using expert judgment as an estimating method.

Overconfidence Example

Olive is assigned to the coding changes for your high-profile project. When asked about the amount of work involved in completing her tasks, she often prefaces her response with phrases like, "This is a piece of cake," and "No problem." Even you, the project manager, are taken aback by the aggressiveness of Olive's schedule. Ironically, the more difficult the task the quicker she plans to get it done. "No problem" might end up being a big problem.

Bias-Reduction Strategies

There are ways to decrease the risk of having an estimate that reflects reality in only the most fortunate of situations. May luck be your constant companion, but just in case, you can:

1. Encourage Olive to develop a range instead of a point estimate. Make sure she considers the worst case, Murphy's Law-type of scenario for the high-end of the range (a pessimistic estimate).
2. Ask Olive on what assumptions this estimate range is based. If the answer is "We will have two fully dedicated clairvoyants developing the requirements," adjust the estimate upward.
3. Gather information from a variety of sources to get a broader picture of what needs to be done.
4. Tell the team that you are not interested in best-case estimates, but realistic estimates. Some studies have shown this will help; some have shown it will not matter, but it cannot hurt.

Remember—studies show our estimates are even more optimistic the more complex and difficult the task [7].

Confirmation Bias

This is in effect when people search for information that supports their idea and ignore information that does not support their idea. An analyst who develops a task-level estimate may consider information supporting the estimate, and ignore information that the task at hand may be significantly larger or smaller than that initial estimate indicates.

Confirmation Example

Cathy is a perfect example. She tends to stick to her guns, and can be narrow-minded. She tends to start estimating her work effort with an estimate in hand and, after limited research, usually concludes she was right in the first place.

Bias-Reduction Strategies

Cathy can be helped. Here is how:

1. Encourage her to research historical data and metrics, and ask other experienced team members for help. It is important to get her to look at

a variety of information, not just data to support her initial estimate.
2. Play devil's advocate. Question her sources and assumptions.
3. Most importantly, ask if she adjusted her initial estimate based on information she found after her research.

Insufficient Anchor Adjustment Bias

This occurs when an initial estimate is made (the anchor), and upon re-estimating the effort, insufficient adjustments are made from that anchor. It does not matter if the initial estimate is derived from historical data, parametric modeling tools, or a random number generator.

Insufficient Anchor Adjustment Example

The task of creating a test plan falls to Alice, who likes to get other team members' opinions on how much effort they think the task entails. She hates a blank sheet of paper. Someone estimates the task at 25 effort hours. After assessesing the available information, she determines that 25 hours seems low, and decides to double the estimate to 50 hours based on limited research. Chances are this adjustment is insufficient, simply because it is based on the initial anchor of 25 hours. The task turns out to require 250 effort hours. This is the danger of the anchoring bias.

Bias-Reduction Strategies

There are methods you can employ as a project manager, or conscientious team member, to try and avoid the negative aspects of the anchor bias:

1. Encourage Alice to research the problem and really dig into it.
2. Ask her to specify a range rather than a point when researching the effort required. This will denote the uncertainty involved and reduce the tendency to insufficiently adjust subsequent estimates. Stating the estimate as about 40 to 80 effort hours is less specific and probably easier to adjust, than an early estimate of 52 hours. It pays to be approximately accurate, rather than precisely inaccurate.
3. Do not ask leading questions when inquiring about an estimate. Avoid saying, "Alice, what do you think? Is this about 50 hours?" Or, "Can we get this done by my birthday next week?" Let Alice do her homework and then negotiate.

Prudence Bias

When faced with high-profile tasks, or the first few times *accountability* is used in the same sentence as *task-level estimates*, team members may respond by coming up with over-cautious estimates [8]. Padding task estimates can be just as dangerous as wildly optimistic low-ball estimates.

Prudence Example

Paul follows all the rules, but you do not want to get behind him on the freeway if you are in a hurry. He takes it pretty slow to be safe, and also pads his task-level estimates to be safe. If several team members follow Paul's lead, the result can be a wildly over-cautious project estimate. Have you heard of of Parkinson's Law? Work expands to fill the amount of time available for completion

Bias-Reduction Strategies

Paul is an asset; he realizes the need to take a closer look at the estimating process. In order to get a more accurate estimate, however, try these techniques:

1. Ask him if he added a cushion or padded his estimate. Pad the project estimate, not each task estimate.
2. Emphasize the need for accurate effort estimates at the task level and show how padding each task will inadvertently lengthen the critical path.
3. Olive the optimist and Paul probably do not have a lot to talk about, but it is a good idea to have them review each other's estimates.

See Table 1 for a summary of the biases that impact software development task-level estimates.

Table 1. *Summary of the Biases and Bias Reduction Strategies*

Bias	Description	Bias Reduction Strategies
Availability	Vivid or graphic events overshadow objectivity	• Challenge the assumptions of the estimate. • Use more than one project/task as a reference. • Discuss the estimate as a team.
Representative	Not using base rate data or metrics	• Use data as well as intuition. • Adjust estimate toward the mean. • Formulate estimate before discussing as a team.
Overconfidence	Too optimistic	• Use an estimate range vs. a point estimate. • Challenge the assumptions. • Use more than one source of information. • Set expectations for realistic estimates.
Confirmation	See what you want to see	• Use historical data and metrics. • Play devil's advocate. • Stress the importance of adjusting the estimate.
Anchor Adjustment	Insufficient adjustment of subsequent estimates	• Foster a research-based estimate. • Use an estimate range vs. a point estimate. • Do not ask leading questions or throw out a guess.
Prudence	Too pessimistic	• Pad the project estimate, not the task estimates. • Discuss the estimate as a team.

General Bias-Reduction Strategies

It is virtually impossible to eliminate the impact of human biases on software project estimating. Biases are by definition subconscious. The same psychological mechanism that creates the bias works to conceal it [4].

The first and most important step is awareness that human biases impact decision-making, particularly decisions with uncertainty—like task-level estimating in software development. If the project team can anticipate, identify, and minimize the negative impact of biases in the software estimating process there will be greater certainty in the validity of the project estimate. General strategies will help reduce the human biases in software project estimates. These strategies are:

- Provide feedback on the accuracy of the estimates.
- Collect data to provide rules of thumb.
- Challenge team members to defend and justify their estimates.
- Emphasize the importance of estimating.
- Use more than one estimating method.

Provide Feedback on the Accuracy of the Estimates

Compare the actual effort hours logged on each task to the current estimate. That way the team member knows where he or she is vs. where he or she should be, and can make adjustments accordingly. Do not discount the team member's perception of what work remains or how far along he or she is, but do not rely on that information alone. It is also a good idea to collect data across several completed projects to use as starting points for future estimates. In addition, be sure to collect the original estimate as well as the actual effort required to complete the task [9].

Collect Data to Provide Rules of Thumb

In addition to the original estimate and actual hours logged, other data are useful to provide a history of a project task. At a minimum collect data related to:

- The source of the estimate (best guess, past projects, etc.).
- The activity driver (number of installs, number of requirements, lines of code).
- The assumptions used (especially skill level, resource dedication, requirements volatility).

Let us take Alice's test plan as an example. She should document where she collected the information for the task-level estimate, the activity driver for the task (perhaps the number of test cases), and her assumptions. This data will be very useful the next time she or anyone else develops a task-level estimate for a test plan. Over the course of many projects, it may be found that given this type of project and testing environment, it takes approximately four hours per test case to complete the test plan. If she estimates she will have about 50 test cases, her initial effort estimate might be around 200 hours. Of course the estimate should be adjusted (and perhaps not insufficiently), but the data collected provides an excellent place to start, and a handy rule of thumb. It also provides a repeatable process, which can be improved upon.

Challenge Team Members to Defend, Justify their Estimates

Estimates are based largely on uncertainty. The more information you can find related to the task at hand, the less uncertainty is involved. Question and challenge the estimates, the source of the data, and the assumptions. The adage of garbage in, garbage out applies here.

Emphasize the Importance of Estimating

To paraphrase President Eisenhower, estimates are nothing, estimating is everything. Discourage the path of least resistance or the permanent sacrifice of accuracy for a temporary reduction in effort. Do not just settle for an estimate, but encourage estimating. The real *expert* in the expert judgment approach is homework, not just experience, and the team needs time to do it right.

Use More Than One Estimating Method

Use a variety of estimating methods and sources of information. Use historical data (if you do not have any, start collecting it), industry statistics, estimating tools, organizational metrics data, experienced team members, best guesses, and even intuition. Comparing multiple estimates lets you know if your team is really getting a handle on the project. For example, it is always a good idea to compare the phase-level estimates from a top-down approach, using a parametric modeling tool, to the aggregated task-level estimates from a bottom-up approach. If they are close, you know you are talking apples and apples.

Imagine being dropped off in a remote location. Being lost is a lot like coming up with an estimate. You are not sure where you are, but you have to know before you can figure out where to go. Imagine you reach in your pocket and find a hand-held global positioning system. Things are looking up. You hit a button and find out where you are. However, that estimate of your location is not based on one satellite (a single source of data); it is based on two or three, as your team's estimates should be. Estimating is like putting together the pieces of a puzzle. There is no *answer*, just indicators that need to be analyzed and managed. See Table 2 for a summary of the general bias-reduction strategies and examples.

Table 2. *Summary of General Bias Reduction Strategies*

General Bias Reduction Strategies	Example
Promote awareness	Talk about the impact of biases on estimates
Provide feedback on the accuracy of estimates	Track and report estimates to actuals for tasks
Collect data to provide "rules of thumb"	Record estimates, actuals, assumptions, and size measures for future reference
Challenge team members to justify their estimates	Document and question assumptions and sources
Use more than one estimating method	Combine task level estimates and compare to phase estimates
Emphasize the importance of estimating	Give team members the opportunity to research their estimates – encourage estimating

Conclusion

Human biases influence and generally have a negative impact on the development of task-level estimates. Although it is impossible to obviate these biases, awareness, understanding, and the incorporation of bias-reduction strategies can help mitigate their negative impact.

We have taken a step back to discuss what we feel to be the root cause of poor task-level estimates using the expert judgment approach during bottom-up estimating. The expert judgment method is viable, and likely to remain one of the most popular methods of developing software project estimates for some time. The next step will be determining to what extent these biases impact software project estimates, and where information technology project managers should focus their efforts to reduce the negative consequences of bias in the software estimating process. Our hope is that the suggestions we have provided here can help you and your team develop better task-level software project estimates.

References

1. Godson, Philip. To Err is Human, to Estimate Divine. *InformationWeek,* January 18 ,1999, pp. 65-72.
2. Kahneman, Daniel, and Tversky, Amos The Framing of Decisions and the Psychology of Choice. *Science,* Vol. 211 No. 30, January 1981, pp. 453-458.
3. Hughes, Robert T. Expert Judgement as an Estimating Method. *Information and Software Technology,* Vol. 38, 1996 pp. 67-75.
4. Jones, Morgan D. *The Thinker's Toolkit,* 1998, Random House, Inc.
5. Simon, Hervert A. *Psychology Review,* Vol. 63, p. 129, 1956.
6 Lim, Lai-Huat., and Benbasat, Izak. A Framework for Addressing Group Judgment Biases with Group Technology. *Journal of Management Information Systems,* Vol. 13, No. 3, Winter 96-97, pp. 7-24.
7. Bazerman, Max. H. *Managerial Decision Making,* 2nd Ed., 1990. John Wiley & Sons, Inc.
8. Hammond, John. S. Keeney, Ralph. L., Raiffa, Howard. The Hidden Traps in Decision Making. *Harvard Business Review,* Sept.-Oct. 1998, pp. 47-58.
9. Demarco, Tom *Controlling Software Projects,* Yourdon Press Computing Series, 1982.

About the Authors

David Peeters is a project manager in the Information Services Division at American Family Insurance in Madison, Wis. He has more than 10 years experience in application development, project planning, and project management. Peeters has a bachelor's degree in computer information systems from Southwest State University and a master's of business administration from Illinois State University.

David Peeters
American Family Insurance
6000 American Parkway
Madison, Wis. 53783
Voice: 608-242-4100 ext. 31348
Fax: 608-243-6558
E-mail: DPeeters@AmFam.com

George Dewey is President of Pathfinder Global Group Inc. He is a certified Project Management Professional, with more than 20 years of experience in corporate planning, project scheduling, cost control, estimating, and management with numerous consulting firms and client organizations. Dewey holds a bachelor's degree in industrial engineering from North Dakota State University and a master's of business administration from Virginia Polytechnic Institute.

George Dewey
Pathfinder Global Group Inc.
5358B N. Lovers Lane #113
Milwaukee, Wis. 5225
Voice: 713-827-4481
Fax: 414-464-3005
E-mail: GDewey_PGGI@MSN.com

About the Authors of Does Calibration Improve Predictive Accuracy?, continued from page 17

Daniel V. Ferens is a corporate affordability officer at the Air Force Research Laboratory at Wright-Patterson AFB in Dayton, Ohio. He is also an adjunct associate professor of software systems management at the Air Force Institute of Technology (AFIT), Graduate School of Logistics and Acquisition Management, at Wright-Patterson AFB, where he teaches courses in software estimation and software management in general. He was the advisor for the 10 theses described in this paper, is an active member of the Society of Cost Estimating and Analysis, and a lifetime member of the International Society of Parametric Analysts. Ferens has a master's degree in electrical engineering from Rensselaer Polytechnic Institute, and a master's degree in business from the University of Northern Colorado.

AFRL/IFSD, Bldg 620
2241 Avionics Circle, Room S1Z19
Wright-Patterson AFB, Ohio 45433-7334
Voice: 937-255-4429, ext. 3558
FAX 937-255-4511
E-mail: daniel.ferens@sn.wpafb.af.mil

David S. Christensen is an associate professor of accounting at Southern Utah University. He was the reader for the 10 theses described in this paper. He received his doctorate degree in accounting from the University of Nebraska-Lincoln in 1987, and has published more than 50 articles in the area of defense cost management. He taught cost management topics at the Air Force Institute of Technology from 1987-97. He is a member of the American Accounting Association, the Institute of Management Accountants, and the Society of Cost Estimating and Analysis.

Southern Utah University
College of Business
351 West Center St.
Cedar City, Utah 84720
Voice: 435-865-8058
Fax: 435-586-5493
E-mail: ChristensenD@suu.edu

Case Study: Automated Materiel Tracking System

The Automated Materiel Tracking System is a Web-based solution created for real-time tracking of more than 1 million materiel pieces transferred between Air Force Materiel Command (AFMC) divisions and the Defense Logistics Agencies at Hill Air Force Base, Utah; Tinker Air Force Base, Oklahoma; and Warner Robbins, Georgia. It works equally well on traditional and wireless local area networks, and was created to replace the present manual data entry and paper-based tracking system, which was labor-intensive, error-prone, and difficult and cumbersome to gather and extract reliable, useful data. The replacement system needed to not only track where and when a material order was placed and delivered, but also act as an efficient data-sharing bridge between intercompany departments. The new system had to provide full-database interfaces over a variety of operating systems and environments and dynamically manipulate legacy and newly gathered data utilizing existing standard office suite software. Data had to be Web-accessible from mid-tier, desktop-level hardware as well as from the hand-held, radio frequency-based computers in the field.

The Automated Materiel Tracking System (AMTS) had to offer end users what technology has always promised: cost-effective, easy, real-time access to a myriad of time-sensitive and detail-specific information. AMTS needed to function as the technology bridge between different operating systems and data structures, and allow access to an incredible amount of data from multiple virtual sites and geographical locations with a seamless interface, so the user need only point and click to obtain needed information. AMTS had to level the playing field by offering a way for simplistic, complex and legacy systems to communicate with each other, allowing end users to quickly and easily extract data as usable information employing various hardware input and output options.

Myron Anderson, OO-ALC/LGN, sponsored the prototype of the AMTS solution, which was successfully implemented in the summer of 1999 at Hill Air Force Base Within days of the prototype deployment, end users were able to provide empirical data on the timeliness of the receipt of the materiel. Collected date and time stamps could prove when an item was ordered, readied for shipping, and delivered to a specific location. Part of the project's success is its ability to access and manipulate Web-enabled information from the desktop, as well as on handheld scanning units used in the field via wireless local area network (LAN) and Internet technologies. AMTS is projected to go online simultaneously at Tinker and Warner Robbins in the first quarter of 2000. Although the project began for Air Force use only, the Navy has expressed interest in implementing AMTS at its depot locations.

AMTS Development Research

The project began as a way to track all materiel movement activities *between* AFMC divisions and a Defense Logistic Agency (DLA). It expanded to track the actual delivery sites within each AFMC division. The AMTS' expanded project scope included Hill Air Force Base with approximately 30 delivery sites, handling approximately 700 transactions daily; Tinker Air Force Base with approximately 100 delivery sites, handling approximately 2,000 transactions daily; and Warner Robbins Air Force Base with approximately 150 delivery sites, handling approximately 3,000 transactions daily.

The original AMTS required distinctions in individual process steps involved in a materiel delivery transaction, including:

- Receipt of the requisition by the DLA's Depot Supply System.
- Material picking process.
- Packing process.
- Transporting and shipping.
- Final receipt of the materiel by the maintenance customer.

The main challenge presented for the project was to find a way to determine and track if and at what time a materiel order was involved in each step of a delivery transaction. Although the materiel shipping and delivery movements were tracked on paper manifests, discrepancies were common regarding the *requested* delivery time and location vs. the *actual* delivery time and location. Delivery urgency specifications (needed within two or four hours, etc.) usually dictated the final transportation cost of an item; the exact time between order acceptance and final delivery needed to be tracked in a way that could satisfy shipping *and* receiving parties of the transaction.

The original paper-based manifest system required data entry of a 14-character alphanumeric string package identification code at each point along the delivery route. This process was very labor-intensive and prone to data entry errors. Quantitative metric data was difficult to compile and, once compiled, not always accurate or reliable. As a solution, hand-held laser bar code scanners became an intrinsic portion of AMTS, significantly reducing data entry errors.

AFMC and DLA staff members identified specific features and benefits:

- Conversion of several legacy data information systems.
- Point-and-click ease of use (for access, query, input, assessment report functions).
- Online help features.
- Standardized, customizable tracking procedures and reporting functions.
- Easy, low complexity updating procedures.
- Lower costs and time required to manually investigate a shipping discrepancy.
- Information protection on the client intranet with client-designated levels of authorization.
- Overall system security (to prevent malicious and accidental breaches).
- Ability to utilize wireless portables for use in tracking purposes and Internet access over a wireless LAN for dynamic data access and manipulation
- Designed to be as lightweight as possible and work on a desktop computer with performance as low as 486/33MHz with no appreciable performance hit.

Additional requested enhancements to the original system included tracking

depot maintenance items returning to the supply system, clearing of in-transit records, tracking of issues from other supply systems, and developing a series of analysis tools to ensure peak performance and continual improvement.

AMTS Development Process

Bar Coding Challenges

Untethered mobility is a major requirement for the project. Having a scanning terminal physically connected to a LAN would severely hamper productivity. In addition to substantially decreasing data entry errors, the portable computers' wireless LAN technology complemented the AMTS solution. And, the wireless capability also allows for Internet access, using any Web browser (i.e. Netscape or Explorer), and allows easy access to AMTS information for immediate use in the field.

To begin a transaction within AMTS, a bar coded manifest is generated from the DLA Depot Supply System. This physical manifest accompanies the materiel order as it flows through individual steps within the shipping and delivery process. At each step in the delivery process, the bar code is scanned and logged into AMTS to track its progress and timing, and the data are then recorded with time and date stamp verifications. The system allows for data capture of the following data elements:

- Item(s).
- Shipping and delivery locations.
- Ordering and delivering parties.
- Receiver of goods.

As data reliability was a major issue, standard and specialized business and logic rules were implemented in the design to ensure data integrity, with data elements routinely checked for appropriateness in fields and string length. Additionally, tables were put in place to either allow or deny a transaction. For example, if an item's destination was scheduled for warehouse A but was delivered to warehouse B, AMTS would produce an appropriate error message so the problem could be investigated and corrected.

Data Structure Challenges

Although the legacy information systems were developed using leading-edge technology at the time, they were originally developed and populated before the desktop PC became available. There were also several versions of systems, each written in its own flavor of database language. Additionally, the legacy reports and query capabilities were very elementary in nature due to the archaic database design structure, and data access was difficult due to mainframe architecture. In essence, AFMC had legacy data only, and it needed searchable, formattable, information, which could be accessed and manipulated in real time, merged with new AMTS data, and used in statistical and accountability reports in standard existing commercial off-the-shelf software, such as Microsoft Access, Excel, and PowerPoint.

Although the AMTS prototype was developed in Access and Excel, releases of the software are also available in two other versions to meet existing standardized environments or preferences:

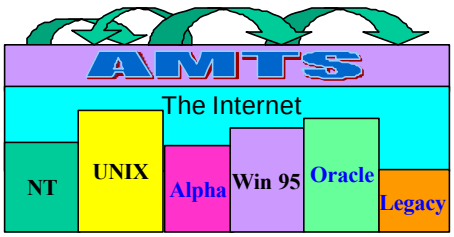
- Oracle 8i with a Visual Basic interface.
- SQL with a Visual Basic interface.

AMTS also uses Microsoft's remote scripting technique to create a dynamic, Web-based user interface. Utilizing their Internet Information Server, Scripting Engines 5.0, and Internet Explorer 5.0, this technique allows for HTML-based Web pages to interact with the server without having to reload the page with each transaction.

Remote scripting uses client-side VBScript, Jscript, or JavaScript on the Web page for data verification and rapid data entry. The client-side script then calls methods through Microsoft's Remote Scripting Jscript Applet that refers to procedures and functions coded in server-side VBScript or JavaScript on an Active Server Page. These procedures and functions interact with server components via .dll files developed in Visual Basic. The .dll files interface with the Oracle 8 database through ODBC connectivity to access, update, and retrieve data. Data are then returned to the .asp page and manipulated further if necessary before being returned to the client-side script on the .htm page where the data can be dynamically displayed without reloading the entire page. The effect is a seamless, active user interface much like a Visual Basic form on an Internet browser.

Due to security concerns with Java and ActiveX applications' open-ended environments, Visual Basic was designated as AMTS' programming language. The main interface of AMTS is Web-enabled; if a user were to exit AMTS to access another site, he or she could introduce a hostile application back into the host network, posing security and data integrity risks. This was a major concern.

Again, AMTS utilizes any Web editor or browser to capture, query, and display information. The ability to use existing, industry-standard software allows for quick, cost-effective use of AMTS at the desktop level or in the field, regardless of database interface.



Bridging the Technology

Not only does AMTS level the playing field and provide full database interface with any operating system environment; it also acts as a technology bridge between intercompany departments or distinct companies or entities.

During beta testing at Hill Air Force Base, AFMC and DLA personnel commented that the user-friendly graphical interface design and online help functions allowed them to sit down and begin using the AMTS software immediately. In just a few days after implementation, end users were able to extract data elements from their legacy systems and combine it with current AMTS data and produce reliability reports using existing off-the-shelf database software.

Conclusion

AMTS is a synthesis of advanced automated data input, proven database technology, and malleability offered by a Web interface. All this is brought together in a single application set that provides realistic information from reams of data. AMTS is available now and is in daily use, supplying timely and reliable information while providing cost-effective and hard empirical metrics.

About the Author

Jim Restel is working with Productive Data Systems on a Web-enabled OO-ALC/LA Supervisor's handbook and AMTS for OOALC/LGN. He is a Defense Infomation Systems Agency Information Warfare Staff Officer, and was the first reservist assigned to Automated Systems Support and Information Security Team (ASSIST)/DoD-Computer Emergency Response Team (CERT). He is a technical advisor to the DoD Joint Web Risk Assessment Cell. Restel has also been a Contingency Plans Staff Officer at the Ogden ALC Readiness Center and a War Plans Officer in Germany and Texas.

Jim Restel, Systems Engineer
Productive Data Solutions
1572 N. Woodland Park Drive, Suite 510
Layton, Utah 84041
Voice: 801-779-2070/
Fax: 801-779-2075
E-mail: jamesrestel@sprintmail.com

Sixth Annual Joint Aerospace Weapon Systems Support, Sensors, and Simulation Symposium and Exhibition (JAWS S³) is scheduled for June 25-30 in San Antonio, Texas

Over the years, this event has addressed target acquisition, the dirty battlefield, the electromagnetic spectrum and its impact on smart and brilliant weapons, and a host of other relevant topics.

This year's conference will feature dialogue up and down the "defense system RDT&E food chain" between the labs and the theaters of operation.

JAWS S³ will focus on the connectivity of various levels of modeling and simulation and their connectivity in support of this mission. JAWS S³ 2000 will feature senior-level decision-makers, who are in a position to impact the directions on these important defense issues, sharing their insights.

— Jim O'Bryon, Deputy Director, Operational Test and Evaluation/Live Fire Testing, Office of the Secretary of Defense

Contact Dr. Asha Varma via electronic mail at varmaa@navair.navy.mil for more information.

Defense@E-Business
April 24-25, 2000
Crystal City Hilton
Arlington, Va.

Distributed Networked Computing for a Secure Defense

The Office of the Secretary of Defense (OSD C3I) is hosting a Distributed Networked Computing Forum called DEFENSE@E-BUSINESS on April 24-25, 2000 in Arlington, Va. at the Crystal City Hilton. DEFENSE@E-BUSINESS is a two-day best practices forum specifically designed for the needs of enterprise system designers, architects, CIOs, and CTOs.

Government IT leaders, integrators, industry practitioners and industry groups will collaborate in sharing lessons learned in developing secure and interoperable frameworks for E-business. You are invited to participate in this annual industry-specific forum and help usher in "Distributed Networked Computing for a Secure Defense."

Confirmed Speakers Include

Dr. V Garber, OSD C3I
Dr. Susan Gragg, ICON
Amy Robinson, Discovery
Paul Kendall, DOJ
Larry Cogut, PTO
General Anthony Bell, Air Force
Tony Scott, GM
Terry Santaviccia, NSA

Registration and Hotel Information

Register online at www.theotg.com
Registration fees are as follows:
Commercial $395
Government $295
Meals are an additional charge of $50
Reserve your room at the Crystal City Hilton by calling (703)418-6800

Conference information is available at www.theotg.com

Requirements Management as a Matter of Communication

Requirements work started as a dialogue between a vendor and an end user. Today this dialogue has been complicated through introduction of marketing and acquisition personnel. This has led to introduction of requirements specifications, but the need for a basic dialogue is still there since a specification cannot capture the increase of knowledge that will always take place during development. An efficient dialogue must not only include requirements, but also stated missions, problem management and understandable formalism for the system's structure and behavior. Further study of the dialogue shows that requirements management must be managed as a process in parallel with processes for development and verification.

Once there was a buyer (end user) who asked a producer, "Can you sell me this and that and what is your price?" Alternatively the producer could ask a buyer, "Do you want to buy this and that at this price?"

This was a long time ago, before the days of complex systems, when both buyers and producers understood the meaning and the use of *this* and *that.*

Today, with complex systems, the situation has become more complicated:

- The buyer is not always the end user since the end user needs help from acquirers and contractors who understand more about bargaining and contract issues than end users' real needs.
- The seller is not always the producer since the producer needs to take help from marketing and sales people, who understand more about marketing than about the product.
- Nobody involved really understands *this* and *that,* because the product sold is a new, complex system, which was not seen until delivery.

The basic questions above are still asked, but the complicated situation makes the answers somewhat hazy, leading to a situation where requirements specifications and management are needed.

Today's Situation

We are in a situation where too often a system project runs like this:

- The end user and the acquirer put together a requirement specification. They get help from one or more consultants to make it complete and correct. The result is a specification, which requires considerable work just to be read and understood.
- The acquirer asks for proposals based on the specification.
- Several vendors look into the specifications, with various reactions, such as:
  - We can do it, but we do not have the time to analyze these requirements until we get the contract proposal of $40 million.
  - This looks interesting, but these requirements need to be analyzed. Advance us $3 million to analyze the requirements and build a first model of the system.
  - We do not understand these require ments, but we desperately need business. We will accept a proposal of $20 million, with 30 percent in advance.
- The acquirer interprets the acquisition regulations and awards the contract to the third vendor as it had the best price.
- A couple of years pass, and the vendor tries to understand the requirements and build a system in accordance with this understanding. During the process of understanding, the vendor will find a number of inconsistencies, contradictions and gaps in the requirements.
- The vendor runs out of money and returns to the acquirer and says, "There are some problems with this contract that need to be sorted out. We need an additional $20 million or we cannot complete it."
- The acquirer is left with two alternatives:
  1. Not paying extra, not getting a system, and possibly causing the vendor to go bankrupt.
  2. Paying extra and getting a system that is probably late and not equal to the true needs since the end users' understanding of needs has grown during system development.

The Need for Dialogue

As previously discussed, there are obvious problems. If you talk to people involved in building military software systems, for example, you will get some clear opinions about what causes the problems:

The military end user: "These keyboard monkeys do not understand a thing about military needs. They do not even understand that you cannot put an M317 backward on an A32!"

The programmer: "These brass hats do not understand a thing about their own systems. They try to express accuracy percentage in inches."

This may seem humorous, but as long as the most important players in system production (the end user and the producer) talk *about* each other instead of *to* each other, the prognosis for the system to be produced is not very good. What is needed is dialogue and . . .

- Understanding—in a language, which is common to everyone involved.
- Respect—for others' professional competence, with no name calling.
- Courage—to speak up whenever someone says or writes something you do not understand.

Take Care of Knowledge Growth

If you start a dialogue between end users and producers in a complex system development project, it is inevitable that knowledge grows among those involved. Knowledge growth may reveal fundamental glitches in the original specification and/or the original proposed solution.

If you have a fixed contract with negotiated deliveries and payments, the best thing about the situation is that it is a real challenge to the contractors, who may not understand there is a problem.

What you need is a work principle, which anticipates that knowledge will grow during system development and that this new knowledge will change the

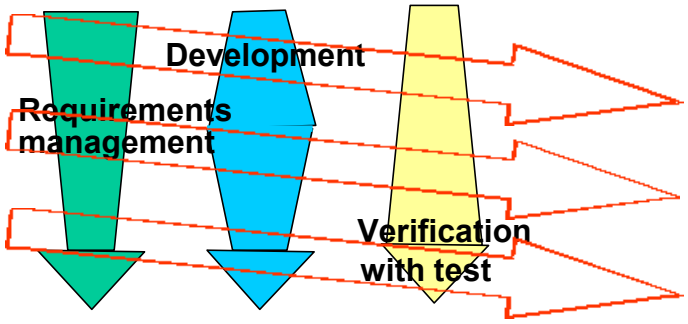
direction and content of the development effort. To find a contract form that considers growth of knowledge is the real challenge for contractors.

Requirements on the Dialogue, Its Language

Besides the obvious need for mutual respect, the dialogue and its language require that:

- The dialogue must use a language that is readily understandable by end users and developers without much training in the language used.
- The language used must be formal in order to avoid misunderstandings.
- The language must be able to describe objects to be managed by the system under development.
- The language must describe system performance precisely.
- For critical systems, the dialogue must support discussion of fault-tolerance issues, including unexpected operator behavior.
- The dialogue must include definition, discussion, and decision on problems, which will surface during development of any nontrivial system.
- The dialogue must anticipate that knowledge will grow during system development and allow this knowledge to influence requirements and design solutions.
- The dialogue must accept that requirements management, development, and verification are three parallel processes during a systems engineering effort.

Figure 1: *Successive Deliveries with Parallel Processes*



Elements of a Solution Alternative

As understood from the aforementioned reasoning, there is more to requirements management than writing and accepting a requirements specification. The following are some aspects to be used as the basis for necessary dialogue.

Missions and Scenarios

To the end user, a system's missions are often so obvious that he does not even state them. Instead, he defines a set of scenarios that may or may not cover the complete mission space. On the other hand, the developer will interpret the specification and create a number of *use cases,* which may or may not cover the mission space.

To create understanding, it is necessary that the end user state the missions clearly as they constitute the foundation of requirements and design. Scenarios can be added to clarify the missions' meaning. The missions are fundamental.

One way to express missions is as *mission objects,* which are supported by other objects used to complete the mission at hand.

Original requirements are often stated in a requirements specification. They may also be found in published standards and regulations. Railway systems, for example, are heavily regulated.

To make it possible to work with the original requirements, you need a database. Many system engineering tools offer such a database where you can insert the requirements by copying and pasting from a document or even having the original document parsed automatically for requirements. One can also use common office tools like Word or Excel to get a low-cost requirements database.

However, doing requirements insertion manually is strongly recommended in order to check uniqueness, testability, contradictions, etc. These checks give valuable understanding of any problems pertaining to the original requirements.

Derived Requirements.

As knowledge grows during development and design decisions are taken, new requirements will surface. These are called derived requirements, and should be put in the requirements database as derived.

Compositive Structure Allocation

Provided the system under development is modeled as a compositive structure, allocation of requirements is simple. A compositive structure is where the system is composed from objects and connected through their interfaces in such a way that it is always obvious which objects depend on other objects to complete their action. The compositive structure makes it possible to float requirements downward through the structure until you find the object, which the requirement should be tested with. The requirement becomes the fulfillment requirement of that object.

Problem Management

Management of problems or issues is a very important part of the dialogue during system evolution. Problems inevitably surface; most of them require a combined effort from end users and developers to find a feasible solution.

This means that the dialogue must include problem management, including:

- Problem statements with category and priority.
- Problem headings and numbers.
- Solution alternatives.
- Solution decisions.

It is possible to manage this with an ordinary word processor, but a tool that supports at least numbering of the problems is preferable.

Understandable Formalism

An end user who has expressed himself in clear English, with diagrams, would normally believe he has made himself completely clear. That will often be the case, but it is amazing how such clear requirements are transformed into software and hardware that does unanticipated things.

It is doubtful that you would try to make the end user write formal specifications, since the necessary knowledge is normally not available when the specification is written.

You could, however, provide a formal representation as part

of the design effort with the objective to:

1. Get a reconfirmation from the end user that the specification is understood in an acceptable way.
2. Give a detailed and formal basis for the implementation work, be it hardware, software, or an operator role.

One way to get this formal description is to use formalized English, a simplified language containing:

- Variables of defined types.
- Control structures.
- Comments.

Test Specifications

Requirements are not much good unless they can be tested. If they are distributed to design objects as fulfillment requirements, you can design test cases for each object to test that they are met.

To ensure a correct set of test cases has been written, they should be reviewed by the end user. It is helpful if test cases are presented together with the requirements they are intended to verify. One way to do this is to produce a requirements/test case matrix.

Conclusions

There are several aspects of requirements management. The most important issue is to establish a dialogue between end users and producers of a system.

It has also been found that the original requirements specification is only part of the information required for successful requirements management.

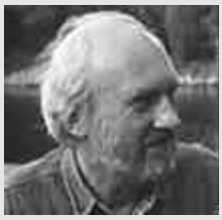
It is a necessity to make requirements documentation understandable both to the end user and to the developers concerned.

Since the dialogue results in more information than can be comfortably managed manually, computer-based support tools will be helpful.

Furthermore, there is a need to create work situations where everyone involved communicates and respects the professional competence of others involved.

Finally, you cannot do the requirements in the beginning of a project, but you must establish a requirements management process in parallel with processes for development and verification as visualized in Figure 1.

About the Author



Ingmar Ogren graduated with an master's degree in science (electronics) from the Royal University of Technology in Stockholm in 1966; his final project was connecting a fighter aircraft training simulator with a sector operation center. He worked with the Swedish Defense Material Administration and various consulting companies until 1989 in systems engineering areas such as communications, aircraft, and command and control. Ogren chairs the board and holds majority ownership in two companies: Tofs, which produces and markets the Tool For Systems software, and Romet, which provides systems engineering consulting with the Objects For Systems method as its main product.

Ingmar Ogren
Romet Corp.
Fridhem2
SE-76040 Veddoe
Sweden
Internet: www.toolforsystems.com

Cost Estimation Web Sites

http://www.computer.org/tse/ts/s1997/e0485abs.htm

This site connects to a paper written by members of the Institute of Electrical and Electronics Engineers. The authors propose the use of a model they say considerably improves early prediction over the Putnam model An analytic proof of the model's improved performance also is demonstrated on simulated data.

http://www.jsc.nasa.gov/bu2/PCEHHTML/pcehfc.htm

This online version of NASA's second edition of the Parametric Cost Estimating Handbook contains seven chapters, and seven appendices, one of which is the parametric estimating system checklist.

http://www.eng.hmc.edu/courses/E180b/software.htm

This is a listing of professional societies and their links relating to software cost estimation. Among the links are the Software Technology Support Center, Defense Information Systems Agency, NASA Software Engineering Laboratory, and the Software Engineering Institute.

http://renoir.csc.ncsu.edu/SP/HTML/cost.html

The North Carolina State University Software Engineering Resource Center site has a tutorial on software cost estimation, and tools.

http://xanadu.bmth.ac.uk/staff/kphalp/students/bsi/predict/tsld002.h tm

This shows slides on software cost estimation, including newer approaches to Boehm's Constructive Cost Model (COCOMO), such as function points and Estimation by Analogy.

http://www.concentricmc.com/toolsreport/5-4-2tools1.html

This tools list on cost estimation has related links to commercial tools, tools used by government departments, and tool reviews and comparisons.

http://www.cpsc.ucalgary.ca/~adi/621/essasy_outl.htm

This site contains an essay on software size estimation, based on references that include the STSC Metrics Service, Watts Humphrey, Capers Jones, and the International Function Point Users Groups.

http://cse.usc.edu/TechRpts/Papers/usccse98-506/usccse98-506.html

This is a report from the Computer Science Department at the University of Southern California's Center for Software Engineering. The report appendices includes a COCOMO II Cost Estimation Questionnaire and COCOMO II Delphi Questionnaire, Defect Removal Model Behavioral Analysis.

http://irb.cs.tu-berlin.de/~zuse/metrids/History_00.html

This is a history of software measurement by Horst Zuse.

http://cse.usc.edu/COCOMOII/cost_bib.html

This is a general bibliography on cost estimation, updated in November.

State of the Software Industry

You've heard the state of the union but what about the state of the software industry? Maybe we should tap the wit and wisdom of those who founded, fathered, and led this nation. Here is my interview with George Washington, John Adams, Benjamin Franklin, Thomas Jefferson, and Abraham Lincoln.

Q: Gentlemen, what are your impressions of this industry we call software?

George: *Software [Discipline] is the soul of an organization [army]. It makes small numbers formidable, procures success to the weak, and esteem to all.*

Abe: *I could as easily bail out the Potomac River with a teaspoon as attend to all the details of software [the army].*

John: *I am well aware of the toil and blood and treasure, that it will cost us to maintain software [this Declaration].*

Q: What are your thoughts on the Microsoft Antitrust Suit?

Thomas: *I am mortified to be told that, in the United States of America, the sale of software [books] can become a subject of inquiry, and criminal inquiry, too.*

Ben: *A nerd [countryman] between two lawyers is like a fish between two cats.*

Q: The Internet revolutionized the way we do business. Why is it so influential?

Thomas: *We hold these truths to be self-evident: that all men are created equal; that they are endowed by their creator with certain unalienable rights; that among these are life, liberty, and the pursuit of the Internet [happiness].*

John: *The Internet [Liberty], according to my metaphysics, is a self-determining power in an intellectual agent. It implies thought, choice and power.*

Q: Do you have any advice for those climbing the ladder to CMM level five?

Ben: *Never confuse motion with action.*

George: *I know the CMM [patriotism] exists, and I know it has done much in the present contest. But a great and lasting software organization [war] can never be supported on this principle alone. It must be aided by a prospect of interest, or some reward.*

Ben: *The CMM [U.S. Constitution] doesn't guarantee success [happiness], only the pursuit of it. You have to catch up with it yourself.*

Q: Why is Dilbert the industry's poster boy?

John: *In every society where software [property] exists there will ever be a struggle between management [rich] and engineer [poor].*

Abe: *Nearly all men can stand adversity, but if you want to test a man's character, give him power.*

Ben: *To be humble to superiors is duty, to equals courtesy, to inferiors noble.*

Q: Why are so many projects over budget and behind schedule?

Abe: *We must not promise what we ought not, lest we be called on to perform what we cannot.*

Ben: *In short, the way to a successful project [wealth], if you desire it, is as plain as the way to market. It depends chiefly on two words, industry and frugality; that is, waste neither time nor money, but make the best use of both.*

Q: What is your advice for software engineers?

Thomas: *Nothing gives one person so much advantage over another as to remain always cool and unruffled under all circumstances.*

Abe: *Always bear in mind that your own resolution to success is more important than any other one thing.*

John: *La molesse est douce, et sa suite est cruelle.*
[Idleness is sweet, and its consequences are cruel.]

– Gary Petersen, Shim Enterprises

Editor's Note: A continuation of this interview will appear in our May issue.

Clarify Your View

Helping Organizations Buy and Build Software Better

through

Process Improvement
Project Management
Systems Engineering
Software Acquisition
Capability Assessments

keeping you informed with

CrossTalk: The Journal of Defense Software Engineering

Software Technology Conference

S T S C

www.stsc.hill.af.mil

Software Technology Support Center

OO-ALC/TISE
7278 4th Street
Hill Air Force Base, Utah 84056

801-775-5555
DSN: 775-5555
Fax: 801-777-8069

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

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

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