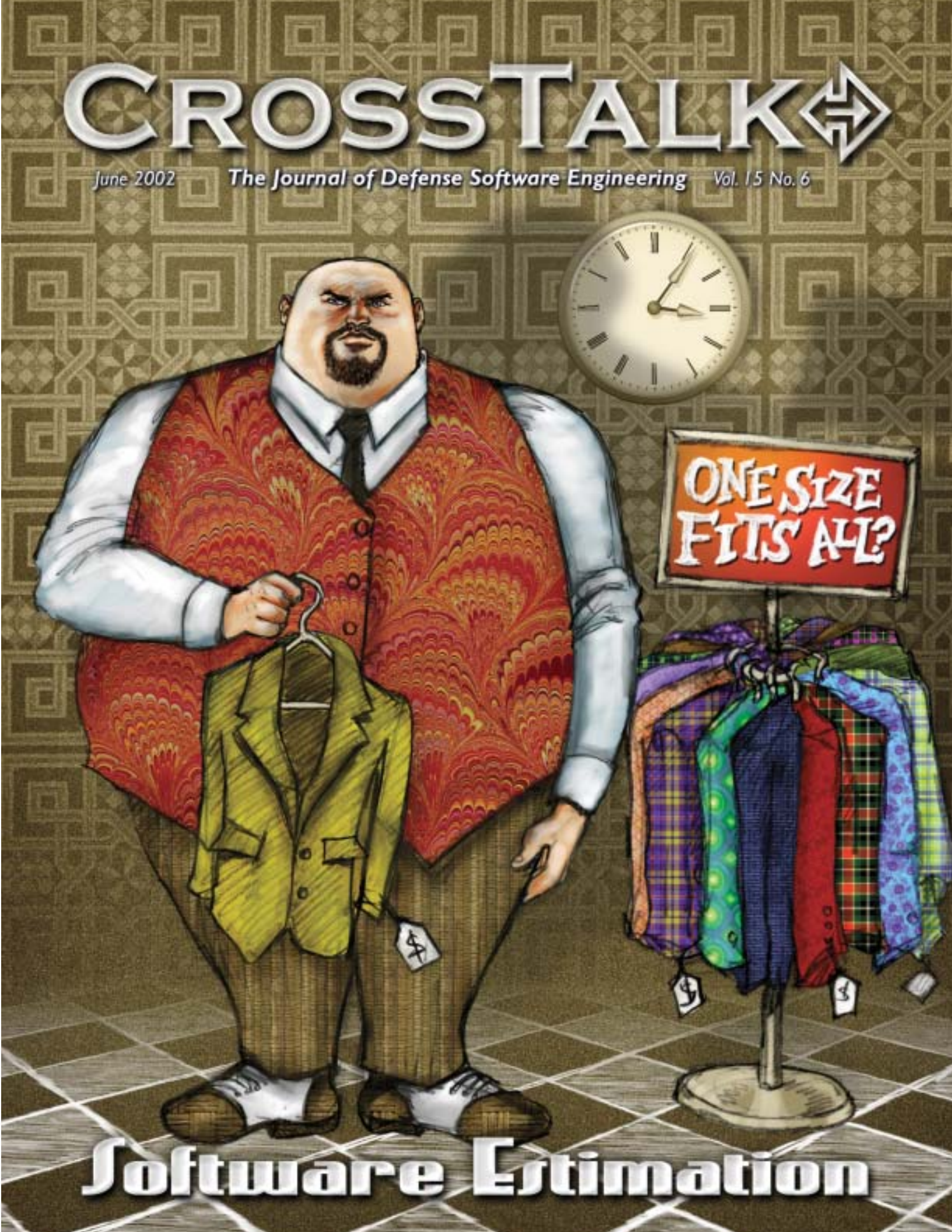
CrossTalk

June 2002 The Journal of Defense Software Engineering Vol. 15 No. 6



Software Estimation

Software Estimation

4 Software Cost Estimation in 2002
This article identifies the major features of software cost estimation tools and how they are being used together technically with project management tools.
by Capers Jones

9 Early Estimation of Software Reliability in Large Telecom Systems
This article describes a number of experiences of early estimation of software reliability for large real-time telecommunications systems.
by Alex Lubashevsky

13 Estimating Web Development Costs: There Are Differences
The challenges in estimating cost and duration in Web development projects prompted one consulting firm to create a new size metric and a new cost estimation model.
by Donald J. Reifer

18 Estimating Software Earlier and More Accurately
The estimating model in this article provides accurate early estimates by utilizing function points to identify project complexity modified by a value adjustment factor of 14 general system characteristics.
by David Garmus and David Herron

ON THE COVER
Cover Design by Kent Bingham.

Best Practices

22 New Code Analyzes Fluid Flow for Better Designed Aerospace Vehicles and Components
The project in this article demonstrates a new way to develop government software using Internet-based tools to leverage costs among organizations.
by Dr. Greg D. Power

Software Engineering Technology

24 Measuring Calculus Integration Formulas Using Function Point Analysis
The size and complexity of algorithms in general and integration formulas in particular can be measured using function points without the need for additional patches or counting rules.
by Nancy Redgate and Dr. Charles Tichenor

Open Forum

28 Software Estimation: Perfect Practice Makes Perfect
Managers looking to initiate a software estimation process in their organization will get practical advice from this author's real-life experiences with estimation training methods.
by David Henry

Departments

3 From the Publisher

26 Coming Events

27 Web Sites

31 BACKTALK

CROSSTALK

SPONSOR	*Lt. Col. Glenn A. Palmer*
PUBLISHER	*Tracy Stauder*
ASSOCIATE PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Pamela Bowers*
ASSOCIATE EDITOR	*Julie B. Jenkins*
ARTICLE COORDINATOR	*Nicole Kentta*
CREATIVE SERVICES COORDINATOR	*Janna Kay Jensen*
PHONE	(801) 586-0095
FAX	(801) 777-8069
E-MAIL	crosstalk.staff@hill.af.mil
CROSSTALK ONLINE	www.stsc.hill.af.mil/crosstalk
CRSIP ONLINE	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 21.

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

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

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

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

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC Online Services: www.stsc.hill.af.mil
Call (801) 777-7026, e-mail: randy.schreifels@hill.af.mil

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

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

Good Software Estimation Requires Historical Data

This month's CROSSTALK focuses on software estimation techniques used within the industry. These articles all agree that good software estimation requires an understanding of the capacity of any particular organization to deliver a software solution within their specific environment. Such capacity evaluation must be based upon historical contexts and good risk management techniques. These articles also provide a good perspective on the state-of-the-art in software estimation and emphasize the importance of measuring how well estimation is being done through each phase of your projects.

Capers Jones' article, *Software Cost Estimation in 2002*, provides a good overview of the types of functionality that are typically present in the approximately 50 commercial software estimation tools marketed in the United States. He covers 10 generic features most often found in many of these tools and conjectures about future trends in software estimation. This article is a good tutorial on the basics of software estimation. Alex Lubashevsky's article, *Early Estimation of Software Reliability in Large Telecom Systems*, describes how using two estimation techniques simultaneously can provide successful results. He used a U.S. Air Force Rome Laboratory model for an early estimation of software reliability, along with a second estimating technique derived from company-wide software process assessments based on the Software Capability Maturity Model® (SW-CMM®).

The revolution of new business strategies employing Web technology has highlighted the need for new ways to measure development time associated with Web projects. In his article, *Estimating Web Development Costs: There Are Differences*, Donald J. Reifer describes a new size metric, Web objects, and a new cost estimation model, WEBMO, that have been developed to satisfy the estimation needs for Web projects. He also discusses some of the key differences that managers must recognize between traditional software development and Web development to be successful in accurately estimating their projects.

The article by David Garmus and David Herron, *Estimating Software Earlier and More Accurately*, is a good synopsis of the function point method of software cost estimation. It claims that utilization of a functional sizing technique such as function points provides the capability to accurately estimate a project early in the development process.

Although not specific to software cost estimation, the article by Dr. Greg D. Power of Sverdrup Technology, Inc., *New Code Analyzes Fluid Flow for Better Designed Aerospace Vehicles and Components*, demonstrates a new way of doing software development for government aerospace applications involving analysis of fluid flow. This article reports on a partnership between the Air Force's Arnold Engineering Development Center and the NASA Glenn Research Center for development of a computational flow simulator.

Nancy Redgate and Dr. Charles Tichenor identify in their article, *Measuring Calculus Integration Formulas Using Function Point Analysis*, specific steps for handling mathematical formulas using function point analysis techniques. Such integration formulas are typically embedded in many engineering and scientific applications. The methodology does not require any new counting rules or patches and promises to give users a more accurate view of application size, resulting in better forecasting of costs, schedule, and quality.

Finally, David Henry gives us some practical lessons learned in his article *Software Estimation: Perfect Practice Makes Perfect*. He focuses on utilizing input from all the engineers involved in a software development project in order to develop personal software estimation accuracy, and on using actual historical performance data to assist in future estimation efforts. He also covers simple methods for group estimation techniques that he has found effective for creation of an organizational estimation process.

I hope that our theme articles will provide a little more insight into your challenges in developing an accurate software estimation capability within your organization. You may want to consider using CROSSTALK to report your organization's successes and lessons learned in cost estimation or any other unique software development capability progress.

Deputy Director, Computer Resources Support Improvement Program

Software Cost Estimation in 2002©

Capers Jones
Software Productivity Research Inc., Artemis Management Systems

The first automated software cost estimation tools were developed independently by researchers in major corporations and military groups in the 1960s. Commercial software cost estimation tools began to be marketed in the 1970s. By 2002, about 50 commercial software cost estimation tools were marketed in the United States and another 25 in Europe. Although standard projects can now be estimated with fairly good accuracy, there are always new technologies that require improvements in estimating tools.

Research on software cost estimation started independently in a number of companies and military organizations that built large software systems. Formal research into software cost estimation became necessary when software applications and systems software began to go beyond 100,000-source code statements in size. This size plateau was reached by several organizations in the 1960s.

The main issue that led to formal research programs for software cost estimation was the difficulty encountered in completing large software applications on time and within budget. A secondary issue was the fact that when deployed, software applications often contained significant numbers of bugs or defects. The evolution of software estimation tools is described in articles by Boehm [1, 2] and Jones [3](each of which describes the state-of-the-art tools at the time of publication). A timeline of the evolution of software estimation tools is shown in Figure 1.

As of 2002, about 50 commercial software estimation tools were marketed in the United States. The major features of commercial software estimation tools include the following basic abilities:

- Sizing logic for specifications, source code, and test cases.
- Phase-level, activity-level, and task-level estimation.
- Support for both function point metrics and the older lines-of-code (LOC) metrics.
- Support for specialized metrics such as object-oriented metrics.
- Support for *backfiring* or conversion between LOC and function points.
- Support for software reusability of various artifacts.
- Support for traditional languages such as COBOL and FORTRAN.
- Support for modern languages such as Java and Visual Basic.
- Quality and reliability estimation.

Additional features found in some but not all software estimation tools include the following:

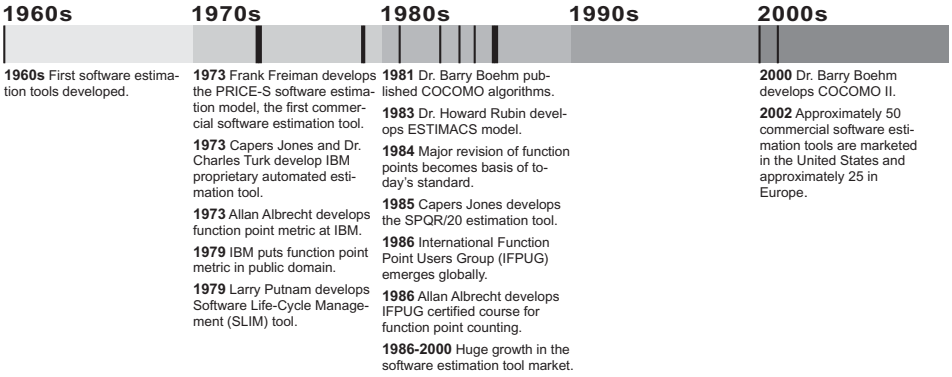
- Risk and value analysis.
- Estimation templates derived from historical data.
- Links to project management tools such as Artemis or Microsoft Project.
- Cost and time-to-complete estimates mixing historical data with projected data.
- Currency conversions for international projects.
- Inflation calculations for long-term projects.
- Estimates keyed to the Software Engineering Institute's Capability Maturity Model® (CMM®).

Modern software cost estimation tools are now capable of serving a variety of important project management functions. However, there are still some topics that are not yet fully supported even by state-of-the-art software estimation tools. Some of the topics that may require manual estimation include the following:

- Conversion and nationalization costs for international projects.
- Fees for trademark and copyright searches.
- Acquisition costs for commercial off-the-shelf packages.
- Deployment costs for enterprise resource planning applications.
- Litigation expenses for breach of contract if a project is late or over budget.

For ordinary software projects, automated estimation tools can now predict more than 95 percent of the associated effort and cost with fairly good accuracy. But projects that must be converted for sale in many countries, or that run on multiple hardware and software platforms, will have expenses outside the scope of most commercial software estimation tools. The legal expenses are also outside their scope if a software project is subject to litigation such as breach of contract or theft of intellectual property.

Figure 1: *Evolution of Software Estimating Tools*



A Large Tool Family

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

The software cost estimation industry and the project management tool industry originated as separate businesses. Project

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

management tools began appearing around the 1960s, about 10 years before software cost estimation tools. Although the two were originally separate businesses, they are now starting to join together technically.

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

Project management tools have no built-in expertise regarding software, as do software cost estimation tools. For example, if you wish to explore the quality and cost impact of an object-oriented programming language such as Smalltalk, a standard project management tool is not the right choice. By contrast, many software cost estimation tools have built-in tables of programming languages and will automatically adjust the estimate based on which language is selected for the application.

Although there are scores of software cost estimation tools on the market, there are 10 generic features that many software estimation tools can perform:

Feature 1: Sizing Specifications, Source Code, and Test Cases

The first step in any software estimate is to predict the sizes of the deliverables that must be constructed. Before about 1985, software cost estimation tools did not include sizing logic. For these older tools, the user had to provide size information. Size data were expressed in LOC for estimation tools developed before the publication of function point metrics.

After function points became available in 1978, size could be expressed using either function points or LOC metrics, and converted between the two. As of 2001, sizing is a standard feature in more than 30 commercial software cost estimation tools.

The advent of function point metrics has eased the burden on software size estimation. Function point totals can be derived from software requirements long before any code is written. Once the function point size of an application is known, then many artifacts can also be sized. These include but are not limited to the following:

1. Specification volumes.
2. Source code volumes.
3. User documentation volumes.
4. Numbers of test cases.
5. Numbers of possible bugs or errors.

Another important sizing aspect is dealing with the rate at which requirements *creep* and hence make projects grow larger during development. If the function point totals for an application are measured at the requirements phase and again at delivery, the two values can be used to calculate the monthly rate of growth.

After the requirements are initially defined, the observed rate of requirements creep is from 1 percent to more than 3 percent per calendar month during the design and coding phases. The average rate of requirements creep is about 2 percent per month based on analysis of several thousand applications during benchmark and baseline studies.

Function points are not the only sizing method available, of course. Some estimation tools also offer templates derived from common kinds of software applications. Many estimation tools allow users to provide their own size data, if they wish, using either LOC metrics or function points or both. Refer to Kan [4] for a discussion of software metrics used in estimation.

In the United States, the function point metric by IBM, and now maintained by the International Function Point Users Group (IFPUG), is most commonly used for software estimates. Version 4.1 of the IFPUG counting rules is assumed in this article [5]. For a discussion of the accuracy of software function point counting, refer to Kemerer [6].

Feature 2: Selecting Project Activities

Once the initial sizes of various deliverables have been approximated, the next step is to determine which specific activities will be carried out for the project being estimated. Activity selection is one of the major areas where software cost estimation tools excel. There are some 25 common activities that might be performed for a software project, but only large military applications will normally perform all 25. For a discussion of activities and how they

Activities Performed	Web Projects	MIS Projects	System Projects	Military Projects
01 Requirements	3%	7.5%	4%	7%
02 Prototyping	10%	2%	2%	2%
03 Architecture		0.5%	1.5%	1%
04 Project plans		1%	2%	1%
05 Initial design		8%	7%	6%
06 Detail design		7%	6%	7%
07 Design reviews			2.5%	1%
08 Coding	25%	20%	20%	16%
09 Reuse acquisition	5%		2%	2%
10 Package purchase		1%	1%	1%
11 Code inspections			1.5%	1%
12 Ind. Verif. & Valid.				1%
13 Configuration mgt.		3%	1%	1.5%
14 Formal integration		2%	2%	1.5%
15 User documentation	5%	7%	10%	10%
16 Unit testing	25%	4%	5%	3%
17 Function testing	17%	6%	5%	5%
18 Integration testing		5%	5%	5%
19 System testing		7%	5%	6%
20 Field testing			1.5%	3%
21 Acceptance testing		5%	1%	3%
22 Independent testing				1%
23 Quality assurance			2%	1%
24 Installation/training		2%	1%	1%
25 Project management	10%	12%	12%	13%
Total	**100%**	**100%**	**100%**	**100%**
Activities Performed	**8**	**18**	**23**	**25**

Table 1: *Software Activity Variations – Percentage of Staff Effort by Activity (Assumes applications of about 1,000 function points in size or larger)*

vary, see Jones [7]. Table 1 (see page 5) illustrates some of the variances in activity patterns for four different types of projects.

Since variations in the activities performed can affect overall costs, schedules, and productivity rates by significant amounts, it is important to match activities to the project being estimated. More than 100 percent differences in work effort have been observed for projects of exactly the same size due to variations in the activities performed. In general, military projects and systems software projects perform more activities than management information systems or Web applications of the same size.

Feature 3: Estimating Staffing Levels and Specialists

Although staffing, effort, costs, and schedules are all important for the final estimate, a typical place to start estimating is with staffing levels. There are significant variations in staffing levels based on team experience, application size, reusable materials, and other factors.

One of the trickier aspects of estimating the staffing for large applications is the fact that sometimes as many as 35 different occupation groups might be working on a large project at the same time. A list of 20 common software occupation groups observed on large software systems is shown in Table 2.

Since each of these specialized occupations may work for only part of a project's life cycle, and since each form of specialization can have very different salary and bonus packages, it is not a trivial task to handle staffing estimates for large software applications when multiple specialists are utilized.

Table 2: *Common Software Occupation Groups*

Common Software Occupation Groups Involved in Large Applications
1. Architects (software/systems)
2. Configuration Control Specialists
3. Cost Estimation Specialists
4. Data Base Administration Specialists
5. Function Point Specialists (certified)
6. Globalization and Nationalization Specialists
7. Graphical User Interface Specialists
8. Integration Specialists
9. Library Specialists (for project libraries)
10. Maintenance Specialists
11. Project Managers
12. Project Planning Specialists
13. Quality Assurance Specialists
14. Systems Analysis Specialists
15. Systems Support Specialists
16. Technical Translation Specialists
17. Technical Writing Specialists
18. Testing Specialists
19. Web Development Specialists
20. Web Page Design Specialists

Feature 4: Estimating Software Work Effort

The term *work effort* defines the amount of human work associated with a project. The amount of effort can be expressed in any desired metric such as work hours, work days, work weeks, work months, or work years. Usually small projects of up to perhaps 1,000 function points utilize hours for expressing effort, but the larger projects in excess of 10,000 function points normally utilize days, weeks, or months as the unit of measure.

For example, in the United States the nominal workweek is five days of eight hours each, or 40 hours total. Yet the number of effective work hours per day is usually only about six due to coffee breaks, staff meetings, etc. The number of workdays per year will vary with vacations and sick leave, but averages about 220 days per year in the United States. However, in Europe vacation periods are longer, while in other countries such as Mexico and Japan vacation periods are shorter than in the United States.

> *"In real life, schedule estimating is one of the most difficult parts of the software estimation process."*

This kind of knowledge can only be determined by accurate measurements of many real software projects. This explains why software estimation vendors are often involved in measurement studies, assessments, and benchmark analysis. Only empirical data derived from thousands of software projects can yield enough information to create accurate estimation algorithms using realistic work patterns. For discussions of how software effort varies in response to a number of factors, refer to Putnam and Myers [8] or Jones [9].

Feature 5: Estimating Software Costs

The fundamental equation for estimating the cost of a software activity is simple in concept, but very tricky in real life:

Effort x (Salary + Burden) = Cost

A basic problem is that software staff compensation levels vary by about a ratio of 3-to-1 in the United States and by more than 10-to-1 when considering global compensation levels for any given job category. For example, here in the United States there are significant ranges in average compensation by industry and also by geographic region. Programmers in a large bank in mid-town Manhattan or San Francisco will average more than $80,000 per year, but programmers in a retail store environment in the rural South might average less than $45,000 per year.

There are also major variations in the burden rates or overhead structures that companies apply in order to recover expenses such as rent, mortgages, taxes, benefits, and the like. The burden rates in the United States can vary from less than 15 percent for small home-based enterprises to more than 300 percent for major corporations. When the variance in basic staff compensation is compounded with the variance in burden rates, the overall cost differences are notable indeed. For a discussion of software cost variations, refer to Jones [10].

Feature 6: Estimating Software Schedules

Estimating software schedules has been a troublesome topic because most large software projects tend to run late. Close analysis of reported schedule errors indicates three root causes for missed schedules: 1) conservative or accurate schedule projections are arbitrarily overruled by clients or senior executives, 2) creeping requirements are not handled proactively, and 3) early quality control is inadequate, and the project runs late when testing begins.

Formal schedule estimation is an area where cost estimation tools and project management tools frequently overlap. Often the cost estimation tool will handle high-level scheduling of the whole project, but the intricate calculations involving dependencies, staff availability, and resource leveling will be done by the project management tool.

A basic equation for estimating the schedule of any given development activity follows:

Effort/Staff = Time Period

Using this general equation, an activity that requires eight person-months of effort and has four people assigned to it can be finished in two calendar months, i.e.:

8 Months/4 People = 2 Calendar Months

In real life, schedule estimating is one of the most difficult parts of the software estimation process. Many highly

complex topics must be dealt with such as the following:

- An activity's dependencies upon previous activities.
- Overlapping or concurrent activities.
- The critical path through the sequence of activities.
- Less than full-time staff availability.
- Number of shifts worked per day.
- Number of effective work hours per shift.
- Paid or unpaid overtime applied to the activity.
- Interruptions such as travel, meetings, training, or illness.
- Number of time zones for projects in multiple cities.

It is at the point of determining software schedules when software cost estimation tools and project management tools come together. The normal mode of operation is that the software cost estimation tool will handle sizing, activity selection, effort estimation, cost estimation, and approximate scheduling by phase or activity. Then the software cost estimation tool will export its results to the project management tool for fine tuning, critical path analysis, and adjusting the details of individual work assignments.

Feature 7: Estimating Defect Potentials

One reason software projects run late and exceed their budgets may be that they have so many bugs they cannot be released to users. A basic fact of software projects is that defect removal is likely to take more time and cost more than any other identifiable cost element.

The fact that software defect levels affect software project costs and schedules is why automated software cost estimation tools often have very powerful and sophisticated quality-estimation capabilities.

Quality estimates use two key metrics derived from empirical observations on hundreds of software projects: *defect potentials* and *defect removal efficiency*. The defect potential of a software project is the total number of defects that are likely to be encountered over the development cycle during the first 90 days of usage. Defect removal efficiency refers to the percentage of defects found and removed by the developers before release to customers.

Based on studies published in the author's book *Applied Software Measurement* [7], the average number of software errors in the United States is about five per function point (Table 3). Note that software defects are found not only in code, but also originate in all of the major software deliverables in the approximate quantities listed in Table 3.

These numbers represent the total number of defects that are found and measured from early software requirements throughout the remainder of the software life cycle. However, knowledge of possible defects is not the complete story. It is also necessary to predict the percentage of possible defects that will be removed before software deployment.

Feature 8: Estimating Defect Removal Efficiency

Many kinds of defect removal operations are available for software projects. The most common types include requirements reviews, design reviews, code inspections, document editing, unit test, function test, regression test, integration test, stress or performance test, system test, external Beta test, and customer acceptance test. In addition, specialized forms of defect removal may also occur such as independent verification and validation, independent tests, audits, and quality assurance reviews and testing.

> *"One reason software projects run late and exceed their budgets may be that they have so many bugs they cannot be released to users."*

In general, most forms of testing are less than 30 percent efficient. That is, each form of testing will find less than 30 percent of the errors that are present when testing begins. Of course a sequence of six test stages such as unit test, function test, regression test, performance test, system test, and external Beta test might top 80 percent in cumulative efficiency.

Formal design and code inspections have the highest defect removal efficiency levels observed. These two inspection methods average more than 65 percent in defect removal efficiency and have topped 85 percent.

Before releasing applications to customers, various reviews, inspections, and testing steps utilized will remove many but not all software defects. The current U.S. average is a defect removal efficiency of about 85 percent, based on studies carried out among the author's client companies and published in *Software Assessments, Benchmarks, and Best Practices* [9], although the top projects approach 99 percent.

U.S. Averages: Defects per Function Point

Defect Origins	Defects per Function Point
Requirements	1.00
Design	1.25
Coding	1.75
Document	0.60
Bad Fixes	0.40
Total	*5.00*

Table 3: *U.S. Averages in Terms of Defects per Function Point (Circa 2001)*

The number and efficiency of defect removal operations have major impacts on schedules, costs, effort, quality, and downstream maintenance. Estimating quality and defect removal are so important that a case can be made that accurate software cost and schedule estimates are not possible unless quality is part of the estimate.

Feature 9: Adjusting Estimates in Response to Technologies

One of the features that separates software estimation tools from project management tools is the way estimation tools deal with software engineering technologies. There are scores of software design methods, hundreds of programming languages, and numerous forms of reviews, inspections, and tests. There are also many levels of experience and expertise on the part of project teams.

Many software estimation tools have built-in assumptions that cover technological topics like the following:

- Requirements gathering methods.
- Specification and design methods.
- Software reusability impacts.
- Programming language or languages used.
- Software inspections.
- Software testing.

Software estimation tools can automatically adjust schedule, staffing, and cost results to match the patterns observed with various technologies. For additional information on such topics refer to Putnam [11], Roetzheim and Beasley [12], and Jones [9].

Feature 10: Estimating Maintenance Costs over Time

In 2001, more than 50 percent of the global software population was engaged in modifying existing applications rather than writing new applications.

Although defect repairs and enhance-

ments are different in many respects, they have one common feature. They both involve modifying an existing application rather than starting from scratch with a new application.

Several metrics are used for maintenance estimation. Two of the more common metrics for maintenance and enhancement estimation include 1) defects repaired per time interval and 2) assignment scopes or quantities of software assigned to one worker.

The *defects repaired per time interval* metric originated within IBM circa 1960. It was discovered that for fixing customer-reported bugs or defects, average values were about eight bugs or defects repaired per staff month. There are reported variances of about 2 to 1 around this average.

The term assignment scope refers to the amount of software one maintenance programmer can keep operational in the normal course of a year, assuming routine defect repairs and minor updates. Assignment scopes are usually expressed in terms of function points and the observed range is from less than 300 function points to more than 5,000 function points with an average of around 1,000 function points.

Future Trends in Software Estimation

Software technologies are evolving rapidly, and software cost estimation tools need constant modifications to stay current. Some future estimating capabilities can be hypothesized from the direction of the overall software industry.

As corporations move toward Internet business models, it is apparent that software cost estimation tools need expanded support for these applications. While the software portions of Internet business applications can be estimated with current tools, the effort devoted to *content* is outside the scope of standard estimates. The word content refers to the images and data that are placed in Web sites.

As data warehouses, data marts, and knowledge repositories extend the capabilities of database technology, it is apparent that database cost estimation lags software cost estimation. As this article is written, there is no *data-point* metric for ascertaining the volumes of data that will reside in a database or data warehouse. Thus, there are no effective estimation methods for the costs of constructing databases or data warehouses or for evaluating data quality.

For companies that are adopting enterprise resource planning (ERP), the time and costs of deployment and tuning are multiyear projects that may involve scores of consultants and hundreds of technical workers. Here too, expanded estimating capabilities are desirable since ERP deployment is outside the scope of many current software cost estimation tools.

Other features that would be useful in the future include value estimation, litigation estimation, and enhanced support for reusable artifacts. Refer to Jones [3], Boehm [2], and Stutzke [13] for additional thoughts on future estimation capabilities.

Conclusions

Software cost estimation is simple in concept, but difficult and complex in reality. The difficulty and complexity required for successful estimates exceed the capabilities of most software project managers. As a result, manual estimates are not sufficient for large applications above roughly 1,000 function points in size.

Commercial software cost estimation tools can often outperform manual human estimates in terms of accuracy and always in terms of speed and cost effectiveness. However, no method of estimation is totally error free. The current *best practice* for software cost estimation is to use a combination of software cost estimation tools coupled with software project management tools, under the careful guidance of experienced software project managers and estimation specialists.

References

1. Boehm, Barry. Software Engineering Economics. Englewood Cliffs, NJ: Prentice Hall, 1981.
2. Boehm, Barry, et al. "Future Trends, Implications in Software Cost Estimation Models." CROSSTALK Apr. 2000: 4-8.
3. Jones, Capers. "Sizing Up Software." Scientific American Magazine Dec. 1998: 74-79.
4. Kan, Stephen H. Metrics and Models in Software Quality Engineering. Reading, Mass.: Addison-Wesley, 1995.
5. International Function Point Users Group. Counting Practices Manual. Release 4.1. Westerville, Ohio: IFPUG, May 1999.
6. Kemerer, C. F. "Reliability of Function Point Measurement – A Field Experiment." Communications of the ACM 36 (1993): 85-97.
7. Jones, Capers. Applied Software Measurement. 2nd ed. New York: McGraw-Hill, 1996.
8. Putnam, Lawrence H., and Ware Myers. Industrial Strength Software – Effective Management Using Measurement. Los Alamitos, Calif.: IEEE Press, 1997.
9. Jones, Capers. Software Assessments, Benchmarks, and Best Practices. Boston, Mass.: Addison Wesley Longman, 2000.
10. Jones, Capers. Estimating Software Costs. New York: McGraw-Hill, 1998.
11. Putnam, Lawrence H. Measures for Excellence – Reliable Software On Time, Within Budget. Englewood Cliffs, N.J.: Yourdon Press - Prentice Hall, 1992.
12. Roetzheim, William H., and Reyna A. Beasley. Best Practices in Software Cost and Schedule Estimation. Saddle River, N.J.: Prentice Hall PTR, 1998.
13. Stutzke, Richard D. "Software Estimation: Challenges and Research." CROSSTALK Apr. 2000: 9-12.

About the Author

Capers Jones is chief scientist emeritus of Artemis Management Systems and Software Productivity Research Inc., Burlington, Mass. Jones is an international consultant on software management topics, a speaker, a seminar leader, and an author. He is also well known for his company's research programs into the following critical software issues: Software Quality: Survey of the State of the Art; Software Process Improvement: Survey of the State of the Art; Software Project Management: Survey of the State of the Art. Formerly, Jones was assistant director of programming technology at the ITT Programming Technology Center in Stratford, Conn. Before that he was at IBM for 12 years. He received the IBM General Product Division's outstanding contribution award for his work in software quality and productivity improvement methods.

Software Productivity Research Inc.
6 Lincoln Knoll Drive
Burlington, MA 01803
Phone: (781) 273-0140
Fax: (781) 273-5176
E-mail: cjones@spr.com

Early Estimation of Software Reliability in Large Telecom Systems

Alex Lubashevsky
Independent Consultant

This article describes early estimation experiences with software reliability of complex real-time telecommunications systems based on size estimation and the process assessment. It emphasizes the importance of estimation for evaluating the feasibility of proposed reliability requirements and providing a rational basis for design and allocation decisions. A number of critical factors for building a reliability estimation model are discussed along with typical sizing processes and estimation tools. The modified U.S. Air Force's Rome Laboratory model was selected as the best practical candidate for the early estimation of software reliability. A data comparison of its results to a traditional estimation model is presented.

Customers rank reliability first on their list of most critical quality attributes, according to a recent survey of the largest customers of complex real-time telecommunications systems and data published in the Army's software metrics newsletter, *Insight* (Spring 2000) [1]. With the cost of some systems exceeding tens or even hundreds of millions of dollars and with a development duration of more than 12 to 18 months, early reliability estimation can significantly contribute to the success (or early rational cancellation) of the project. With recent strong emphasis on speed of development, the decisions made on the basis of early reliability estimation can have the greatest impact on schedules and cost of software projects. Software reliability may also be significantly improved in an early stage by a focused review and inspection process, early defect removal, and thorough test effort.

Software reliability estimation provides a solid foundation to perform meaningful tradeoff studies at project start. It also provides a projection of the software failure rate before systems tests begin or at any point throughout. After estimation, the next logical step is creating a software reliability growth model, which covers the period where reliability improves as the result of thorough peer reviews, testing, and fault correction. Reliability metrics help to predict such critical factors as the initial failure rate, final failure rate, fault density, fault profile, etc.

The final outcomes of software reliability estimation include the following:

- An estimation of the number of faults expected during each phase of the life cycle.
- A constant failure rate estimation at system release.
- Relative measures for practical use and management such as duration of system test and size of the test team.

If software reliability estimation is performed early in the software life cycle, it is possible to determine what improvement, if any, can be made to the software methods, techniques, or organizational structure. As described in this article, our experience also confirmed what many recent articles and publications have suggested: A successful, meaningful estimating strategy must simultaneously use more than one estimating technique [2].

> *"Software reliability estimation provides a solid foundation to perform meaningful tradeoff studies at project start."*

For early software reliability estimations, an estimation team used a number of software reliability models. The team compared the results of these models with each other and with the reliability data provided by cost/effort estimation model KnowledgePLAN, which was selected a few years ago after Bell Labs' studies of more than a dozen different estimation models (the COCOMO tool came out a very close second). This expert system tool contains in its database more than 8,000 actual projects that along with size, cost, effort, and other attributes have data about the total number of inherent faults, their distribution among major phases of life cycle, the severity of faults, and potential defect removal efficiency [3].

As with any other existing cost estimation tool, KnowledgePLAN had to be fine-tuned and thoroughly calibrated for the particular project to be consistent and reliable. At the same time in this environment, the KnowledgePLAN questionnaires on the project/process development activities were used as a *shell* for the company-wide software process assessments, together with some parts of the Software Engineering Institute's Capability Maturity Model® (CMM®). These assessments were used for internal benchmarking of current development practices and helped to easily identify similar projects, which was essential for a meaningful comparison [4].

Estimating Parameters

AT&T/Lucent's experience in software reliability estimation for a number of large telecommunications projects is evidence that the size of the project (in function points or sometimes in thousands of lines of code [KLOC]) is the most significant single factor for estimating the number of inherent faults [4]. The second most important factor is project complexity, which can be represented by McCabe's cyclomatic complexity measure (problem, code, and data complexity). Those complexity measures strongly depend on the application types of the project [3, 5].

The KnowledgePLAN tool uses simplified McCabe's complexity factors in its questionnaire, as does its predecessor Checkpoint. Also our previous independent studies found that the fault-reduction factor is relatively stable across different projects in the same organization, though the fault-exposure ratio may be dependent on the structure of the program and the degree to which faults are data dependent. However, these effects are often averaged out for programs of substantial size such as the large projects the estimation team often deals with [6].

To increase estimation accuracy and our ability to better control the discovery-of-faults process, the team concluded the following: The software reliability estimation must be performed in early phases of the life cycle by using phase-based models that emphasize the availability of size and

corresponding effort for the project during early phases. In our case, the estimation team obtained the size and effort in the early stage of development very often by using the function point method [7]. However, for some legacy projects, the data were derived by the analogy method or by experts' iterative estimations such as Delphi analysis [5].

Also, the number of factors needed for building reliability estimation, which significantly relates to fault density at the early stages such as application type, development environment, and some other software metrics, can be obtained relatively easily from the results of previously performed company-wide software process assessments. After a number of studies and experiments with different types of reliability models, the team selected the U.S. Air Force's Rome Laboratory model [8]. The estimation team chose this model because it allows us to track the influence on software reliability of the various application types by different development organizations, methods, tools, techniques, and other software factors, and it is closely correlated to our development methodology.

Estimating Size and Reliability Early

Size estimation consists of two phases, called *passes*, which require using highly trained and experienced estimators. After a list of new or modified features is prioritized during the proposal stage, the first pass provides a quick rough estimate of size/effort required to develop the features. These are then given to product management to determine the budget for an upcoming release.

The estimate is determined by breaking up the software into smaller pieces, consulting with experts, and forming analogies to previously developed software with similar features. If a separate estimation repository database updated on a quarterly basis contains the data for a similar feature, the preliminary data for the size/effort estimation and often the total number of inherent faults are readily available. If data from the repository are not available, one of the reliability estimation models with some of the default parameters is used. One or two members of the planning group usually generate the first-pass estimate with accuracy within 50 percent.

The second pass is a detailed estimate done near the end of the requirements process to define size/effort of a feature, a subsystem, or system level. The development planning group usually coordinates the entire process. For some projects, the second-pass estimates are based on the judgment of experienced developers (rather than expert estimators) who are using a bottom-up estimation technique based on historical data to outline and estimate the tasks by functional area that will be required to develop a feature.

This group meeting is used to obtain a single estimate from a group of experts after an open discussion. The meeting is a forum for experts to discuss requirements and designs, consider a tradeoff between the reliability of the product and cost and schedules of the project, resolve issues, and work together to create and tune their estimates. The Delphi technique [5] can be combined with the group meeting approach. The group meeting is used to discuss estimation issues and the experts give their opinions. These estimates are discussed again, and the process is repeated until a consensus is reached.

"Size estimation consists of two phases, called passes, which require using highly trained and experienced estimators."

Size/effort estimation and early reliability estimation by analogy was the most popular method. The technique assumes that if two projects are alike in some respects, other similarities can be inferred. The current project is compared with similar completed projects to get a ballpark estimate; experts then factor in the differences between the projects. Estimation by analogy can be applied to a total system, a module, or a task. A historical database is the best tool for estimation by analogy. Here are sources for analog data in the order of decreasing preferences:

- Data from a previous release of the same project.
- Data from a similar project in the same company.
- Data from a similar project in a different company.

Industry data can be used if company data are not available, but they must be calibrated to company data as soon as they are available.

For a completely new project that may have no relevant data or experience on which to base estimates, particularly if the project is moving to a new methodology such as from a traditional to an object-oriented approach, the following estimation strategy is recommended: Create, if possible, an analog by dividing the product into components and actually doing development on a typical component to estimate the remainder of the project. Also, the following additional information associated with the software cost/effort estimation and reliability estimation is recommended:

- Project management tracks baseline, current and completion dates, and the number of detected faults.
- At the end of each software release, postmortem compares actual vs. detailed estimates for the features.
- The estimation process is evaluated on two criteria: responsiveness (in business days) and accuracy, which is defined as the percent of relative error between the detailed estimate and actual data.

The new estimation process described in the next section also consists of two phases (sometimes an additional zero phase is added) and is based on more consistent usage of estimation tools and techniques and less reliance on high human expertise. But this estimation practice strongly concurs with the suggestion that the most important factor in improving estimation is to "hold its software estimators, developers, and managers accountable for their estimates" [9].

A New Estimation Process

Our new estimation process is based on size estimation using function point analysis (FPA), which is usually performed based on complete requirements [7] in combination with estimation tools like KnowledgePLAN and the modified Rome Labs estimation model. (In some cases, FPA is based on particular telecom domains or, even earlier, based on the high-level Feature Definition and Assessment Form (FDAF), the most important estimation phase called zero pass.)

The project size in function points (FP) is one of the significant inputs for the KnowledgePLAN tool. Other input information collected by the KnowledgePLAN's questionnaire describes the type, nature, and complexity of the project, the project management practices, the expertise and morale of the team, together with a few dozen other attributes of the particular project. This enables the tool to choose the project having the closest match from its vast knowledge base of previously collected industry projects.

As output, the tool generates many

useful estimation reports on resources, schedule, etc., and also on the total number of defects that will be introduced during various stages of the project. While in the past the estimation team widely used the predicted reliability data on potential defects from the Checkpoint estimation tool (KnowledgePLAN's predecessor), the team decided to compare the estimation tool's results with those from another software reliability model.

For this purpose, the original U.S. Air Force's Rome Laboratory model, RL-TR-92-52 [8] was modified to allow for more than 60 telecommunications applications to be included in the historical database, and also to allow for the usage of the FP method, which is growing in popularity in civil and military applications [4]. Using available industry and internal data from the software process assessment (SPA) and the CMM of the organization developing the software, the major nine factors of the Rome Lab's model were expanded to include new ranges of values in FPs. (See Table 1).

Originally, the output of the Rome Lab's model is a fault density in terms of faults per KLOC. To compute the total estimated number of inherent defects, the fault density should be multiplied by the total predicted number of KLOC. If function points are being used and no KLOC is available for correlation, the backfire method (low accuracy table for the conversion of source lines of code to FPs [3, 4]) is sometimes recommended. Also the Rome Lab's model is very useful for predicting fault density at delivery time; subsequently, this fault density is utilized to predict the total number of inherent faults and the failure rate.

The fault density of the application (A) is predicted by using a baseline fault density established for applications of the same type, adjusted to reflect the influence of the development environment (D) and the software characteristics (S) of the specific application. Once fault density is determined, the failure rate (FR) can be predicted by applying empirical value (EV), established from historical data for the application type, to the fault density. The Rome Lab's model contains empirical data that have a total of 33 data sources representing 59 different projects (some from Software Engineering Laboratory).

Fault Density: FD = A x D x S
(faults/FPs or LOC)

Estimated number of Inherent Faults:
N = FD x SIZE

Factor	Measure	Range of Values: Rome Labs defs/KLOC	Range of Values: Telecom defs/FPs	Application Phase*	Trade-off Range
A-Application	Difficulty in developing various application types	2 to 14	0.2 to 1.5	AP-T	None fixed
D-Development environment	Development org., methods, tools, techniques, document	.5 to 2.0	0.1 to 1.8	If known at AP, DTLD-T	Largest range
SA-Software anomaly mgmt.	Indication of fault-tolerant design	.9 to 1.1	0.3 to 0.4	Normally, C-T	Small
ST-Software traceability	Traceability of design and code to requests	.9 to 1.0	0.2 to 0. 6	Normally, C-T	Large
SQ-Software quality	Adherence to coding standards	1.0 to 1.1	0.2 to 0.3	Normally, C-T	Small
SL-Software language	Normalizes fault density by language type	N/A	N/A	C-T	N/A
SX -Software complexity	Unit complexity	.8 to 1.5	0.1 to 0.6	C-T	Large
SM-Software modularity	Unit size	.9 to 2.0	0.1 to 0.7	C-T	Large
SR-Software standards review	Compliance with design rules	.75 to 1.5	0.2 to 0.4	C-T	Large

*AP = Concept or Analysis Phase, C = Coding, DTLD = Detailed and Top Level Design, and T = Testing

Table 1: *Summary of the Rome Laboratory Model, RL-TR-92-52*

Failure Rate: FR = FD x EV
(faults/time)

This model has the following significant benefits:

- It can be used as soon as the software concept is known.
- During the concept phase, it allows *what-if* analysis to be performed to determine the impact of the development environment on fault density. (In our case, the data from the previous SPA for this organization will be reused.)
- During the concept phase, it allows *what-if* analysis to be performed to determine the impact of software characteristics on fault density. (Also the data from the previous SPA for this organization will be reused.)
- It allows for system software reliability allocation because it can be applied uniquely to each application type comprising a large software system.
- The estimation can be easily customized using unique values for the A, D, and SIZE factors based upon historical software data from the specific organization's environment.

The Rome Lab's model consists of nine factors (Table 1, first column) that are used to predict the fault density of the software application. That is, application type factor (which could be real-time control systems, scientific, or information management) with the range of values of two to 14 defects per KLOC, or 0.2 to 1.5 defects per FP. This demonstrates the potential influence of different application types on fault density in an early phase of development. Similar logic applies to the rest of the factors: They show the potential ranges of values for fault density that depend on the type of factors and measures associated with them.

There are parameters in this estimation model that have tradeoff capability (maximum/minimum predicted values). The analyst can determine where some changes can be made in the software engineering process or product to achieve improved fault-density estimation. This tradeoff is valuable only if the analyst has the capability to impact the software development process. (Notice that the tradeoff is fixed for the type of application and is not applicable after you select a particular software language.) The tradeoff analysis can also be used to perform a cost analysis by optimizing the development.

The values of many of the parameters in this model may change as development proceeds. The latest updated values should be used when making an estimation that will become more and more accurate with each successive phase until the final design and implementation.

Table 1 represents the summary of Rome Laboratory's estimation model. The column "Range of Values" shows original and modified telecom values, the latter reflecting the historical reliability data correlated with the data from the SPA assessments range for more than 60 projects. Most of the factors of the original model like software implementation metrics (SX, SM, SR, etc.), requirements and design representation metrics (SA, ST, SQ), and application (A) and development environment (D) correspond almost one-to-one to the factors of the SPA/SPR questionnaire. The only difference is that range of values for fault density is mapped in defects per FPs instead

Severity Level	Predicted	Industry Standard	Actual	Percent of Variance (Actual to Predicted/Industry Standard)
1. System inoperative	87	103	67	22/35
2. Major functions incorrect	408	516	359	14/30
3. Minor functions incorrect	1185	1446	1076	10/25
4. Superficial error	945	1034	742	27/28
Total	2626	3098	2244	17/28

Table 2: *Faults by Severity Level*

of defects per KLOC.

Table 2 provides an example of data for one of the critical telecom projects. The predicted and industry standard data were generated using the project quality report of KnowledgePLAN. Actual data came from the final systems/integration test reports. The data from the two different models were compared and correlated with the SPA data for this project and the reliability data produced by the modified Rome Labs model. The percent of variance between actual and predicted faults showed an acceptable range of deviation (from 10 percent to 27 percent, as compared with the acceptable range for this most important early stage of estimation: +/- 50 percent).

Table 2 represents data for a project size of about 1,800 FPs. The fault density (1.24 faults/FP) and total number of inherent faults (2,626) were predicted for the FDAF stage of the life cycle. These data, together with some historical information for similar projects, helped predict the duration of systems test with about 15 percent accuracy and the size of the test team with about 20 percent accuracy. The early predicted faults in Table 2 are presented by severity level (based on historical distribution data for similar projects) and percent of deviation from actual and industry database. The 17 percent deviation of total predicted faults to actual is a significant (almost three times) improvement compared with the first-pass results (50 percent) at the same stage.

Also during this object-oriented project for a real-time telecommunication system, the early reliability estimation data indicated a need for a focused review and inspection process especially during analysis, design, and additional systems test effort to fully cover all test cases written against original user requirements. This helped to increase defect removal efficiency to 92 percent (which is very high for this type of transmission application) and to create a system with software availability exceeding Telcordia standards by more than 10 times. This particular system was in the field for more than one year without any major software outage.

Summary

This article has described a number of experiences of early estimation of software reliability for large real-time telecommunications systems. The benefits of early reliability estimation for the design and allocation decisions, which can have a significant impact on schedule and cost, were also discussed. The article also emphasized the importance of early size and complexity estimation on which a number of software reliability models are based. The past processes of early size and reliability estimation (relying on highly qualified human experts) and new processes (based on the heavy usage of estimation tools) were described in detail. The modified Rome Labs estimation model, based on early size estimation as well as a number of other factors that describe the software development process and its influence on reliability, was introduced for comparison and was shown to be very useful. The example of working with two different models for early reliability estimation and the positive results achieved proved that other projects could significantly benefit from the above-described processes in building reliable systems.

References

1. Insight 4:1. Spring 2000. (Insight is the Army's Software Metrics Newsletter. Available at: <www.ArmySoftware Metrics.org>.)
2. Shepperd, M. J., and C. Schofield. "Estimating Software Project Effort Using Analogies." IEEE Trans. Software Engineering 23.12 (1997).
3. Jones, T.C. Applied Software Measurement. McGraw-Hill, 1991.
4. Lubashevsky, Alex, and L. Bernstein. "Living with Function Points at AT&T." CROSSTALK Nov./Dec. 1995.
5. Boehm, B. W. Software Engineering Economics. Prentice Hall, 1981.
6. Musa, J. D. Software Reliability. Measurement, Estimation, Application. McGraw-Hill, 1987.
7. Albrecht, A. J., and J. R. Gaffney. "Software Function, Source Lines of Code, and Development Effort Estimation: A Software Science Validation." IEEE Trans. Software Engineering 9.6 (1983).
8. Rome Laboratory. "Methodology for Software Reliability Estimation and Assessment." Technical Report RL-TR-92-52, Vol. 1 and 2, 1992.
9. Lederer A. L., and J. Prasad. "A Causal Model for Software Cost Estimating Error." IEEE Trans. Software Engineering 24.2 (1998).

About the Author

Alex Lubashevsky is an independent consultant specializing in estimation and reliability. He was previously an estimation project manager with AT&T/Lucent for 17 years. He was responsible for estimating size, effort, interval, and defects for more than 200 telecom and data processing systems inside and outside the company (IRS, DELTA, Prudential, etc). While at AT&T, he also helped to achieve one of the first industry Capability Maturity Model® Level 3 ratings. He originated and helped to develop the first automated CASE tool for early estimation (Bachman Analyzer). Lubashevsky was an early industry supporter of Practical Software and Systems Measurement and a pioneer in the National Software Council and later a board member. He has a master's of science degree in computer science from New York University and a bachelor's of science degree in computer science from the Polytechnic University of Kharkov, Ukraine. He is a member of the International Electrical and Electronics Engineers Computer and Communications Societies, the International Function Point Users Group, and the International Academy of the Information Sciences.

165 Osprey
Hackettstown, NJ 07840
Phone: (908) 813-0208
Fax: (908) 813-0208
E-mail: alexluba1@att.net

Estimating Web Development Costs: There Are Differences

Donald J. Reifer
Reifer Consultants, Inc.

This article discusses the need for new metrics and models to estimate the effort and duration for Web development projects. It then describes a new size metric, Web objects, and a new cost estimation model, WEBMO, that have been developed to satisfy these needs. Most importantly, this article identifies differences between traditional and Web projects that managers need to be aware of when developing estimates prior to ending with a current status of the effort.

This past year, electronic commerce reportedly reached $5 billion in sales. Considering that this was during a recession, it is a marvelous achievement. You are probably thinking, "How was that achieved with the technology bubble bursting and Internet start-ups failing right and left?" The answer is simple. The larger businesses took the place of the smaller businesses. They moved to the Web with speed and enthusiasm, often for good business reasons. For example, General Electric reportedly saved $40 million in a single month compared with the same month in the previous year by moving its travel onto the Web [1].

Just as importantly, this move to the Web is heralding in a major change in the way we in the software community do business. For example, the large projects that we worked on in the past are being replaced by many small Web developments. These small projects are being done using different technology as well. Table 1 characterizes these changes to give some insight into the current trends. It highlights the move to agile methods [2], extreme programming methods [3], components [4], multimedia, and visual technologies by Web shops.

These trends are motivated by the move to quicker-paced developments. Instead of developing software from requirements, these Web development projects are gluing components together using agile instead of traditional methods. They build prototypes and refactor [5] them instead of focusing on design. From Table 1, you will see that Web developments seem deficient in the areas of process, discipline, and estimating. That is not entirely true. As Mark Paulk recently pointed out, process improvement and extreme methods are not incompatible [9]. However, many of the large firms with which my firm has recently worked seem to have abandoned process paradigms and the Software Engineering Institute's Capability Maturity Model® (CMM®), Software Capability Maturity Model® (SW-CMM®), and Capability Maturity Model Integration℠ (CMMI℠) [10, 11] in their quest to speed time-to-market as they move to the Web.

Those of us in the estimating community currently have not agreed upon how to address Web-based projects. The trouble is that the characteristics of the Web-based projects listed in Table 1 make it difficult for estimators to adapt and put their existing processes, metrics, and models to work operationally. Web projects are different. To highlight the challenges involved in the area of Web estimation, we have constructed Table 2 (see page 14). For comparative purposes, this table also identifies the approaches that we currently use to develop estimates for traditional software projects.

Most estimators would like to use the more traditional processes, metrics, models, and tools for estimating Web projects. They are mature, and many of us in the field have confidence in their ability to accurately predict project costs and schedules. We also have a great deal of experience using these metrics, models, and tools and feel comfortable with them and their outputs. However, as noted by Table 2 (page 14), these traditional approaches do not address the challenges that we face with Web projects. The two major challenges are accurately estimating size and duration. New size metrics are needed to cope with Web objects like shopping carts, Java scripts, and building blocks like Cookies, ActiveX controls, and Component Object Model components. New duration-estimating equations are needed to address the fact that the cube root laws used by most estimating models just do not seem to work for the Web.

New Web Applications Sizing Metrics Needed

Because Web cost can be treated as a function of size, a meaningful size predictor is needed for Web projects. Those working such projects agree that the popular size metrics, function points (FP) and source lines of code (SLOC), are not suitable for Web estimation because they do not take all of the Web objects (buttons, templates, etc.) into account. Luckily, the research community has not been idle. It has proposed several size metrics for Web developments (object points [12], application points [13], etc.). However, the only find-

Table 1: *Characteristics of Traditional vs. Web Development Projects*

Characteristics	Traditional Developments	Web Developments
Primary objective	Build quality software products at minimum cost.	Bring quality products to market as quickly as possible.
Typical project size	Medium to large (hundreds of team members).	Small (3-5 team members the norm, 30 the largest).
Typical timeline	12-18 months	3-6 months
Typical cost	$ millions	$ thousands
Development approach employed	Classical, requirements-based, phased and/or incremental delivery, use cases, documentation driven.	Agile methods, extreme programming, building block-based, demo-driven, prototyping, Rational Unified Process [6], MBASE [7].
Primary engineering technologies used	Object-oriented methods, generators, modern programming languages (C++), CASE tools, etc.	Component-based methods, 4th and 5th generation languages (html, Java, etc.) visualization (motion, animation), etc.
Processes employed	CMM-based	Ad hoc
Products developed	Code-based systems, mostly new, some reuse, many external interfaces, often-complex applications.	Object-based systems, many reusable components (shopping carts, etc.), few external interfaces, relatively simple.
People involved	Professional software engineers with 5+ years of experience.	Graphic designers, less-experienced software engineers, Java specialists.
Estimating technologies used	Source line of code or function point-based models, Web-based systems approach for small projects.	Design-to-fit based on available resources, Web-based systems for small projects.

Note: Table is an adaptation of the author's previously published work [8].

® Software Capability Maturity Model and SW-CMM are registered in the U.S. Patent and Trademark Office.
℠ CMM Integration and CMMI are service marks of Carnegie Mellon University.

	Traditional Approach	Web-Based Challenges
Estimating process	Most use analogy supplemented by lessons learned from past experience.	Job costing, if done, performed ad hoc using inputs from the developers (too optimistic).
Size estimation	Because systems are built to requirements, SLOC or function points are used. Separate models are used for COTS and reused software (equivalent new lines).	Applications are built using templates using a variety of Web-based objects (html, applets, components, etc.). No agreement on size measure reached yet within the community.
Effort estimation	Effort is estimated via regression formulas modified by cost drivers (plot project data to develop relationships between variables).	Effort is estimated via analogy using job costing practices and experience as the guide. Little history is available.
Schedule estimation	Schedule is estimated using a cube root relationship with effort.	Schedules estimated using cube root relations are an order of magnitude high.
Model calibration	Measurements from past projects are used to calibrate models to improve accuracy.	Measurements from past projects are used to identify estimating knowledge base.
What if analysis	Estimating models are used to perform quantitative *what if* and risk analysis. They are also used to compute ROI and cost/benefits.	Most *what if* and risk analysis is mostly qualitative because models don't exist. ROI and cost/benefit analysis for electronic commerce remains an open challenge.

Note: Table is an adaptation of the author's previously published work [8].
SLOC = source lines of code, COTS = commercial off-the-shelf, ROI = return on investment

Table 2: *Web-Based Estimation Challenges*

ing that researchers in the field currently agree upon is that they cannot reach agreement on which of these is best. Based upon our experimentation, we believe that we have developed a size metric that can resolve the current debate. The metric, Web objects, predicts size by taking each of the many elements that make up the Web application into account as size is estimated.

You are probably asking, "What are Web objects?" Like function points, Web objects are defined to be a metric that provides users with an indication of the relative size of an application [14]. In our case, the applications run on the Web. Web objects predict size by permitting its users to bound the functionality of their applications in terms of the following five groups of predictors:

- Function points.
- Links.
- Multimedia files.
- Scripts.
- Web building blocks.

As indicated, Web objects extend function points to encompass groups of functions present in Web applications. For example, Web objects allow estimators to take Web building blocks like shopping carts and the number of XML language lines needed to link the application to a Web accessible database into account as they develop their estimate. Such extension is needed because traditional function points predict size using more traditional application characteristics like number of inputs and outputs.

Using the size predictor groupings listed in Table 3 to compute the number of Web objects, we have been able to repeatedly predict the size of a Web application with what we believe to be reasonable accuracy. These predictors allow us to take into account all of the different elements of applications that contribute to size, including those specific to the Web. We devised this list initially based upon the opinions of experts. For one year, we have applied the metric to develop estimates, collect project data, and refine our counting conventions based upon experience. Based upon analysis of 64 completed Web projects in five application domains, we have shown that these predictors can be used along with FPs to develop accurate estimates.

Table 3: *Web Object Predictors*

Web Object Predictors	Description
Number of function points	Traditional metrics used to predict the size of non-Web applications using number of inputs, outputs, files, inquiries, and interfaces as the basis of estimate.
Number of XML, HTML, and query language links	Takes into account the effort to link applications, integrate them together dynamically, and bind them to the database and other applications in a persistent manner.
Number of multimedia files	Takes into account the effort required to insert audio, video, and images into applications.
Number of scripts	Takes into account the effort required to link HTML/XML data with applications and files and generate reports.
Number of Web building blocks	Takes into account the effort required to develop Web-enabled fine-grained building block libraries and related wrapper code needed to instantiate them.

Like function points, the key to developing repeatable and consistent sizing predictor counts is a well-defined set of counting conventions. To help on our pilot projects, we have developed a white paper to explain our initial counting conventions [15]. We plan to update this and develop a more detailed counting manual for Web objects later this year. That manual, a version of which will posted on our Web site at <www.reifer.com>, will provide those interested in using Web objects with a consistent set of experience-based conventions for dealing with most situations they will likely encounter when sizing their Web applications.

We have also developed the worksheet in Table 4 to show you how to use the information gathered on predictors along with function points to size a typical Web application. Using an actual Java program being developed for a Web portal as an example, the size estimate developed in Table 4 provides you with an indication of how big the program would be once fully developed. It uses the five groupings of predictors that we have discussed to develop weighted counts that allow us to size a Web application based upon its unique characteristics.

We have also empirically developed backfiring ratios to convert from Web objects to SLOC. If the example shown in Table 4 were done in Java, the 356 Web objects would be the equivalent of 11,392 source lines of Java code, assuming a conversion ratio of 32 Java lines per Web object. Backfiring is important to us because we plan to use a modified version of the COCOMO II early design model to estimate effort and duration.

The COCOMO II model uses SLOC as its underlying size metric. However, after much experimentation, we calibrated our new cost estimation model WEB model (WEBMO) using Web objects instead of SLOC. The reason for this was that such calibration improved our estimating accuracy by as much as 12 percent in two of our five application domains. However, to remain compatible with COCOMO II until we can calibrate all five domains, we plan to continue to use SLOC in our formulas. This may change in the future once we gather more data and can more precisely calibrate our estimating model.

A New Estimation Model

Having a realistic size metric is just the first step in developing a model for accurately

estimating Web application effort and duration. The mathematical issues associated with making such predictions need to be reconciled before such models are transitioned into use.

The major issue revolves around the schedule law used by the model. Analysis of data we have collected to date confirms that the equations can be expressed as regressions. However, the traditional cube-root relationship that exists between effort and duration in most estimation models does not accurately predict Web development schedules [16].

Dr. Barry Boehm of the University of Southern California has proposed a square-root relationship for small projects [17]. Larry Putnam has published several papers arguing that such relationships are accurately represented by a fourth power tradeoff law [18, 19]. Our initial data analysis reveals that a square-root relationship exists for Web projects. However, this mathematical relationship tends to breakdown when the number of Web objects exceeds 300. Therefore, the schedule law used in our model needs to be scaled accordingly.

To estimate Web project costs, we have developed the WEBMO cost model; its mathematical formulation is shown in Figure 1. As stated, this model is an extension of the COCOMO II early design model [20]. The WEBMO model was developed using a mix of expert judgment and actual data from 64 projects using linear regression techniques. It allows users to take the characteristics of Web projects identified in Table 1 into account via adjustments that they make to its cost drivers.

WEBMO's mathematical formulation builds on the COCOMO II model's extensive data analysis of more than 161 projects to address Web issues. We compute exponents for its effort and duration equations, P1 and P2, using the following five application domains: Web-based electronic commerce, financial/trading applications, business-to-business applications, Web portals, and information utilities. As shown in Figure 1, the WEBMO estimating equations for effort (in person-months) and duration (in calendar months) assume size is provided in Web objects. To predict duration, the model assumes a square root instead of a cube-root relationship between duration and effort for small projects.

The current version of the WEBMO estimation model differs from the original COCOMO II model by having nine instead of seven cost drivers and a fixed instead of a variable effort power law.

Web Object Predictors	Low	Average	High	Notes
Traditional function point predictors				
• Internal logical files		2x10	1x15	From specification: 3 files
• External interface files		2x7		From specification: 2 interfaces
• External inputs		4x4	6x6	From specification: 10 inputs
• External outputs		3x5		From specification: 3 outputs
• External inquires				From specification: none
Number of XML, HTML, and query lines		16x4		From specification: 16 HTML lines
Number of multimedia files	1x4 3x4	13x5	1x7	Operands: audio file, 13 multimedia files, help file Operators: open, close, save
Number of scripts	3x2	1x3		Operands: animation script Operators: open, go (forward), close
Number of Web building blocks	3x3	10x4	5x6	Operands: 15 building blocks from library including 9 buttons, 1 cast, and 5 secure server icons Operators: find, add, and insert
TOTAL	**31**	**237**	**88**	

Table 4: *Web Object Calculation Worksheet*

Application Domain	A	B	P1	P2
Web-based electronic commerce	2.3	2.0	1.03	0.5 or 0.32
Financial/trading applications	2.7	2.2	1.05	0.5 or 0.32
Business-to-business applications	2.0	1.5	1.00	0.5 or 0.32
Web-based portals	2.1	1.8	1.00	0.5 or 0.32
Web-based information utilities	2.1	2.0	1.00	0.5 or 0.32

Table 5: *WEBMO Parametric Values*

While our goal is to be as compatible with COCOMO II as possible, we had to deviate because of observed colinearity between cost drivers when we performed our regression analysis. Such colinearity means that some of the cost drivers can not be assumed to be independent from others. In response, we have treated them and COCOMO's scale factors differently in our mathematical formulation.

The constants in the effort and duration equations and power laws for each of the five application domains that we have studied are summarized in Table 5. A brief explanation for each of the nine cost drivers used by the model is provided in Table 6 (see page 16). The values for the driver ratings used in the model are also provided in Table 6. Those interested in more detail on the model are referred to the WEBMO model definition manual that will be issued late this year. At that time, a version of this manual will also be made available at <www.reifer.com>.

As expected, the choice of value for the duration power law, P2, is based on the relative size of the application. For small applications less than 300 Web objects, the square-root relationship between effort and duration seems to hold (e.g., P2 = 0.5). For larger Web applications, the cube-root relationship should be used (P2 = 0.32).

The nine cost drivers replace those in the original COCOMO II model. Most represent combinations of the original factors in the early design version of the model. However, teamwork and process efficiency are new and different. They represent scale factors in the COCOMO II model that have been shown to have a statistical effect on Web estimation. However, instead of using them as power law factors in the effort estimating equation, we include them as effort multipliers to simplify the mathematics. Once we can verify the WEBMO calibration statistically using scale factors, we will revert back to the more standard version of the COCOMO II model.

Let us continue with the Java example we used previously for sizing a Web application. The 356 Web object count in Table 4 represents the size of the program that would be required for this Web application. If this were done entirely using the Java language, the program would take the equivalent of 11,392 SLOC to develop, test, and transition into operations using the Language Expansion

Figure 1: *WEBMO Estimation Equations*

$$\text{Effort} = A \prod_{i=1}^{8} cd_i \, (\text{Size})^{P1} \qquad \text{Duration} = B(\text{Effort})^{P2}$$

Where: A and B = constants; P1 and P2 = power laws; cd_i = cost drivers; Size = # SLOC

Cost Driver	Ratings				
	Very Low	Low	Nominal	High	Very High
Product Reliability and Complexity (CPLX)	Client only, simple math and I/O, no distribution, reliability not a factor.	Client/server, some math, file management, limited distribution, easy to recover.	Client/server, full distribution, databases, integration, moderate recovery goals.	Client/server, wide distribution, math intensive, high losses due to errors.	Client/server, full distribution, collaborative, soft real-time, errors dangerous.
Values	**0.63**	**0.85**	**1.0**	**1.30**	**1.67**
Platform Difficulty (PDIF)	Rare platform changes, speedy net, no resource limitations.	Few platform changes, fast net, few resource problems.	Stable platform, net performance all right, must watch resource usage.	Platform often changes, slow, lack of resources a problem.	Platform unstable, poor performance, resources limited.
Values	**0.75**	**0.87**	**1.00**	**1.21**	**1.41**
Personnel Capabilities (PERS)	15th percentile, major delays due to turnover.	35th percentile, minor delays due to turnover.	55th percentile, few delays due to turnover.	75th percentile, rare delays due to turnover.	90th percentile, no delays due to turnover.
Values	**1.55**	**1.35**	**1.00**	**0.75**	**0.58**
Personnel Experience (PREX)	≤ 2 months, limited tool, language, and platform experience.	≤ 6 months, some tool, language, and platform experience.	≤ 1 year, average tool, language, and platform experience.	≤ 3 years, above average tool, language, and platform experience.	≤ 6 years, lots of tool, language, and platform experience.
Values	**1.35**	**1.19**	**1.00**	**0.87**	**0.71**
Facilities (FCIL)	International, no collaboration, language tools.	Multisite, some collaboration, basic CASE, some methods.	One complex, teams, life-cycle methods, good tools.	Same building, teamwork, integrated tools and methods.	Co-located, integrated collaborative method/tools, etc.
Values	**1.35**	**1.13**	**1.00**	**0.85**	**0.68**
Schedule Constraints (SCED*)	Must shorten, 75% of nominal value.	Must shorten, 85% of nominal value.	Keep as is, nominal value.	Can relax some, 120% of nominal value.	Can extend, 140% of nominal value.
Values	**1.35**	**1.15**	**1.00**	**1.05**	**1.10**
Degree of Planned Reuse (RUSE)	Not used.	Not used.	Unplanned reuse.	Planned reuse of component libraries.	Systematic reuse based on architecture.
Values	**--**	**--**	**1.00**	**1.25**	**1.48**
Teamwork (TEAM)	No shared vision, no team cohesion.	Little shared vision, marginally effective teams and teamwork.	Some shared vision, functional teams.	Considerable shared vision, strong team cohesion.	Extensive shared vision, exceptional team cohesion.
Values	**1.45**	**1.31**	**1.00**	**0.75**	**0.62**
Process Efficiency (PEFF)	Ad hoc, rely on heroes.	Project-based process, rely on leadership.	Streamlined process, rely on process.	Efficient process, best way to do job.	Effective process, people want to use it.
Values	**1.35**	**1.20**	**1.00**	**0.85**	**0.65**

*Schedule differs from COCOMO II, which is bell shaped instead of flat past its nominal value.

Table 6: *WEBMO Cost Drivers and Their Values*

Factors (LEF) listed in Table 7. Please note that the values in this table differ from those currently endorsed by the International Function Point Users Group (IFPUG). We developed these numbers empirically using our Web applications database because the IFPUG numbers did not seem to be consistent with our current experience. This count includes the volume of work required to program Java scripts and beans on both the client and server, assuming that an appropriate Java environment were available for this distributed application.

The LEF factors are used to backfire between FP and SLOC estimates. The one convention that we impose on backfiring is that only languages in the same family can be used in conjunction with each other. For example, you would not mix C and C++ counts because their syntax and semantics are quite different. Our data indicate that there is a 10 percent to 40 percent error in counting when languages are mixed across language families. As noted, HTML and System Query Language (SQL) are considered 4GL and can be mixed using the conventions that we developed.

Why is backfiring to SLOC important? The COCOMO II model uses SLOC as the basis for all of its estimates. Therefore, conversion to SLOC is required to use this popular model out of the box.

Let us run WEBMO with all of its drivers set to nominal within the Web portal domain as an example. The effort estimate assuming size is 11.4 thousand SLOC is about 24 person-months, while the duration estimate is 5.0 calendar months assuming the cube-root relationship holds because size is greater than 300 Web objects.

For comparison purposes, let us run the early design version of the COCOMO II model out of the box with all of its cost drivers set to nominal. However, to use the model, we need to calibrate the scale drivers. We will assume the following values for these parameters:

- Precedence – largely familiar.
- Development flexibility – general goals.
- Architecture/risk resolution – often (60 percent).
- Team cohesion – basically cooperative.
- Process maturity – level 1 (upper half).

With these values, COCOMO II estimates the effort at 38.8 person-months during a period of 11.4 calendar months. "Which estimate is right?" you are probably asking. As expected, neither answer is right on the mark. The actual for this project was six people for four months. Clearly, the WEBMO formulas have better predictive accuracy for this project.

Summary and Significant Research Findings

Estimating the cost and duration of Web developments has a number of challenges associated with it. To cope with these challenges, we developed new size metrics, Web objects, and an estimating model, WEBMO. We have also validated and calibrated the metric and model in anticipation of building potential products based upon them.

We prepared an initial calibration for WEBMO by combining expert opinion and actual data from 64 completed Web projects. Our goal is to improve the accuracy of our models by collecting data on at least another 30 projects during Phase II.

The following significant results/findings were outputs of our initial research efforts:

- We validated that Web objects have better predictive accuracy (r^2) than tra-

ditional function points when counted using conventions developed for that purpose. These counting conventions allowed us to extend the excellent work done by the IFPUG so that we can better handle the sizing of Web applications.

- We increased the statistical accuracy of our WEBMO estimating model from 30 percent of the actual experience at least 60 percent of the time (using a 32-project database of actuals) to 20 percent of the actual experience at least 68 percent of the time (using our expanded 64-project database of actuals).
- We validated that a square root instead of a cube-root relationship exists between effort and schedule for Web application projects whose size was less than 300 Web objects.

These results are substantial because they indicate that the Web objects and the WEBMO estimating model can help address the gaps in the estimating technology that we summarized in Tables 1 and 2.

Acknowledgment

A major part of the research reported within this paper was funded by the National Science Foundation under Grant DMI-0060006. Our partners also provided the data used to calibrate the model and refine its mathematical formulation.

Currently, we are looking for beta test sites and partners for our research. Anyone interested in participating in this capacity can inquire through e-mail at <info@reifer.com>.

Information on COCOMO II

For those unfamiliar with the COCOMO II, refer to the University of Southern California (USC) Web site at <http://sunset.usc.edu/research>. USC has much literature and a public version of the model available on this site.

References

1. Pelz, James P. "GE Takes to the Net to Lower Company Costs." Los Angeles Times 9 Oct. 2000: C1-C5.
2. Highsmith, Jim, and Alistar Cockburn. "Agile Software Development: The Business of Innovation." IEEE Computer Nov. 2001: 120-122.
3. Beck, Kent. Extreme Programming Explained. Addison-Wesley, 2000.
4. Heineman, George T., and William T. Councill. Component-Based Software Engineering. Addison-Wesley, 2001.
5. Fowler, Martin, Kent Beck, John Brant, William Opdyke, and Don Roberts. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999.
6. Kruchten, Philippe. The Rational Unified Process. Addison-Wesley, 1998.
7. Boehm, Barry W. "Transitioning to the CMMI via MBASE." Southern California SPIN Meeting Presentation. University of Southern California, Jan. 2000. Available at: <http://sunset.usc.edu>.
8. Reifer, Donald J. "Web Development: Estimating Quick-to-Market Software." IEEE Software Nov./Dec. 2000: 57-64.
9. Paulk, Mark C., "Extreme Programming from a CMM Perspective." IEEE Software Nov./Dec. 2001: 19-26.
10. Paulk, Mark C., Charles V. Weber, Bill Curtis, and Mary Beth Chrisis. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.
11. Ahern, Dennis M., Aaron Clouse, and Richard Turner. CMMI Distilled. Addison-Wesley, 2001.
12. Lorenz, Mark, and Jeff Kidd. Object-Oriented Software Metrics. Prentice Hall, 1994.
13. Boehm, Barry W., Chris Abts, A. Winsor Brown, et al. Software Cost Estimation with COCOMO II. Prentice Hall, 2000: 192-6.
14. Dreger, J. Brian. Function Point Analysis. Prentice Hall, 1989: 5.
15. Reifer, Donald J. "Web Objects Counting Conventions." Reifer Consultants, Mar. 2001. Available at: <www.info@reifer.com>.
16. Brown, A. W. "CORADMO." 13th International COCOMO Forum and Focused Workshop on COCOMO II Extensions. Oct. 1998.
17. Boehm, Barry W. "COCOMO II Overview." 14th International COCOMO Forum. Oct. 1999.
18. Putnam, L. H. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem." IEEE Trans. Software Engineering. SE-4 July (1978): 345-61.
19. Putnam, L. H., and D. T. Putnam. "A Data Verification of the Software Fourth Power Tradeoff Law." International Society of Parametric Analysts Conference. 1984.
20. Boehm, Barry W., Chris Abts, A. Winsor Brown, et al. Software Cost Estimation with COCOMO II. Prentice Hall, 2000: 51-55.

Language	LEF
1GL default	320
C	128
2GL default	107
COBOL (ANSI85)	91
FORTRAN 107	107
PASCAL	91
3GL default	80
C++	53
Java for Web	32
LISP	64
ORACLE	38
Visual Basic	40
Visual C++	34
Web default — visual languages	35
OO default	29
EIFFEL	20
PERL	22
Smalltalk	20
Web default — OO languages	25
4GL default	20
Crystal Reports	20
Program generator default	16
HTML	15
SQL for Web	10
Spreadsheet default	6
Excel	6
Screen Painter	6
5GL default	5
XML	6
MATHCAD	5

Table 7: *Language Expansion Factors (LEF)*

About the Author

Donald J. Reifer is one of the leading figures in the fields of software engineering and management, with more than 30 years of progressive experience in government and industry. In that time, he has served as chief of the Ada Joint Program Office and the director of the Department of Defense Software Reuse Initiative. He is currently the president of Reifer Consultants, Inc., which specializes in helping clients improve the way they do business. Reifer's many honors include the American Institute of Aeronautics and Astronautics Software Engineering Award, the Secretary of Defense's Medal for Outstanding Public Service, the NASA Distinguished Service Medal, the Frieman Award, and the Hughes Aircraft Fellowship.

P.O. Box 4046
Torrance, CA 90505
Phone: (310) 530-4493
Fax: (310) 530-4297
E-mail: d.reifer@ieee.org

Estimating Software Earlier and More Accurately

David Garmus and David Herron
The David Consulting Group

Software practitioners are frequently challenged to provide early and accurate software project estimates. A U.S. government study on software development projects revealed the extent of that challenge: 60 percent of projects were behind schedule, 50 percent of projects were over cost, and 45 percent of delivered projects were unusable. This article explores the use of a basic estimating model, which utilizes functional sizing as one of its key components. The primary value gained from utilizing a functional sizing technique such as function points is the capability to accurately estimate a project early in the development process.

Two issues are at the core of the estimation challenge. First, is the need to understand and define (as early as possible) the problem domain. Second, is the need to understand the capacity to deliver the required software solution within a specified environment. Then and only then can accurate predictions be made of the effort necessary to deliver the required product.

The problem domain can be defined as the definition of requirements. The problem domain must be accurately assessed in size and complexity. Furthermore, the ability to develop an initial estimate (early in the system's life cycle) must exist since we cannot presume to have all of the necessary information at our disposal. Therefore, a rigorous process must exist that permits further clarification of the problem domain as additional knowledge of the required solution is gained.

The capability to deliver is derived from an assessment of risk factors that are known to impact rate of delivery.

An effective estimation model considers three elements: functional size, complexity, and risk factors. When factored together, the opportunity to achieve an accurate estimate is significantly increased (see Figure 1).

Functional Size

By far, the project-sizing technique that delivers the greatest accuracy and flexibility is function point analysis. Based upon several recent analytical studies performed for client organizations, the function point sizing method was compared with other sizing techniques, including backfiring from source lines of code, approximation, ratios, and estimation. The results concluded that the function point method consistently produced more accurate sizing of the software product.

As to its flexibility, the function point methodology presents the opportunity to size a user requirement regardless of the level of detail available. An accurate function point size can be determined from the detailed information included in a logical, user-defined requirements document or from the limited information provided in an early proposal.

> *"An effective estimation model considers three elements: functional size, complexity, and risk factors."*

The function point method is dependent upon the identification of five elements: inputs, outputs, inquiries, internal stores of data, and external references to data. During the early stages of development, these elements are exposed at a functional level. (For example, we know that we will generate an output report, although we may not know the detailed characteristics of that report.)

The first level of function point counting is to identify these five elements. As more information becomes available regarding the characteristics of these elements such as data fields, file types, and so on, the function point count will become more detailed. During the early phases of a count, it may be necessary to assume levels of complexity within the system (for example, whether the report will be simple or complex). The value of using function points is that it allows for this distinction and, in fact, requires it early in the process.

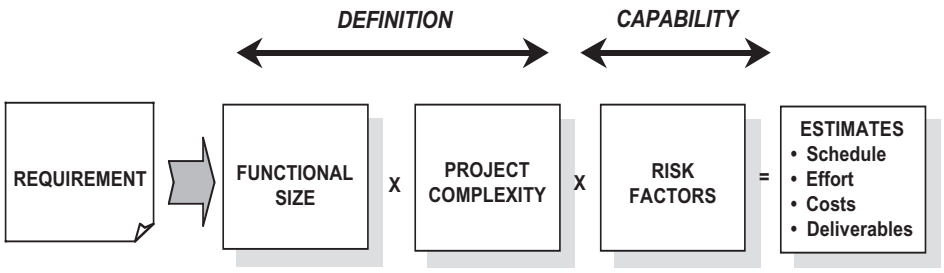
Function points accurately size the stated requirement. If the problem domain is not clearly or fully defined, the project will not be properly sized. When there are missing, brief, or vague requirements, a simple process using basic diagramming techniques with the requesting user can be executed to more fully define the requirements. Function points can be utilized early in the life cycle in conjunction with a context diagram or other diagramming devices such as mind maps or use case diagrams. The developed diagram is used to identify stated inputs, outputs, inquiries, internal stores of data, and external stores of data (see Figure 2). For an average-size project, hours (not days) are required to complete the diagramming and sizing task.

From the example in Figure 2, we can quickly identify, at a high level, the inputs, outputs, internal, and external files. At this level, we could easily assign average values of complexity and quickly determine a functional value that we would then use in our estimating model.

Project Complexity

The second element addressed in our estimation model is project complexity.

Figure 1: *An Effective Estimation Model*



Project complexity must be properly evaluated and should consider the impact of various contributing characteristics that may influence the ease or difficulty in developing the required solution. In part, complexity level is evaluated as part of the function point methodology.

A value adjustment factor (VAF) is used as a multiplier of the unadjusted function point count to calculate the adjusted function point count of an application. The VAF is determined by identifying 14 general system characteristics (GSCs). Each characteristic has an associated description that helps in determining the degree of influence (of that characteristic). The degree of influence for each characteristic ranges on a scale from zero (no influence) to five (strong influence). The 14 GSCs are totaled to calculate a total degree of influence (TDI).

A VAF is calculated from the following formula: VAF = (TDI * 0.01) + 0.65. When applied, the VAF adjusts the unadjusted function point count by ±35 percent to produce the adjusted function point count. Detailed guidance is contained within the International Function Point Users Group (IFPUG) Counting Practices Manual. Information on IFPUG can be found on their Web site at <www.ifpug.org>.

Each of the following 14 GSCs is evaluated and assigned a degree of influence between zero and five:

1. Data Communications: Describes the degree to which the application communicates directly with the processor.
2. Distributed Data Processing: Describes the degree to which the application transfers data among components of the application.
3. Performance: Describes the degree to which response time and throughput performance considerations influenced the application development.
4. Heavily Used Configuration: Describes the degree to which computer resource restrictions influenced the development of the application.
5. Transaction Rate: Describes the degree to which the rate of business transactions influenced the development of the application.
6. Online Data Entry: Describes the degree to which data are entered through interactive transactions.
7. End-User Efficiency: Describes the degree of consideration for human factors and ease of use for the user of the application measured.

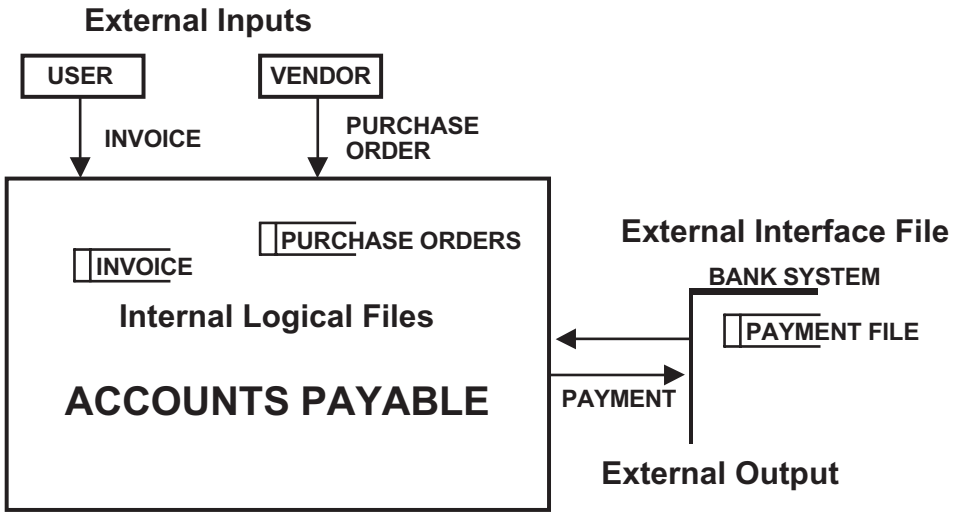


Figure 2: *Example of a Function Point Diagramming Device*

8. Online Update: Describes the degree to which internal logical files are updated online.
9. Complex Processing: Describes the degree to which processing logic influenced the development of the application.
10. Reusability: Describes the degree to which the application and the code in the application have been specifically designed, developed, and supported to be usable in other applications.
11. Installation Ease: Describes the degree to which conversion from previous environments influenced the development of the application.
12. Operational Ease: Describes the degree to which the application attends to operational aspects such as start-up, back-up, and recovery processes.
13. Multiple Sites: Describes the degree to which the application has been developed for multiple locations and user organizations.
14. Facilitate Change: Describes the degree to which the application has been developed for easy modification of processing logic or data structure.

In addition to the 14 GSCs, a project's complexity assessment must take into consideration complex interfaces, database structures, and contained algorithms. These complexity factors impact the project delivery, and they do not serve as an adjustment to the function point count. The complexity assessment can be based upon five varying levels of complexity:

Level 1:
- Simple addition/subtraction.
- Simple logical algorithms.
- Simple data relationships.

Level 2:
- Many calculations, including multiplication/division in series.
- More complex, nested algorithms.
- Multidimensional data relationships.

Level 3:
- Significant number of calculations typically contained in payroll/actuarial/rating/scheduling applications.
- Complex, nested algorithms.
- Multidimensional and relational data relationships with a significant number of attributive and associative relationships.

Level 4:
- Differential equations typical.
- Fuzzy logic.
- Extremely complex, logical, and mathematical algorithms typically seen in military/telecommunications/real-time/automated process control/navigation systems.
- Extremely complex data.

Level 5:
- Online, continuously available, critically timed.
- Event-driven outputs that occur simultaneously with inputs.
- Buffer area or queue determines processing priorities.
- Memory, timing, and communication constraints.
- The most advanced applications developed.

Risk Factors

The capability to effectively deliver software on time and within budget is based upon a variety of risk factors. The third element in our estimating model is an evaluation of risk factors, including soft-

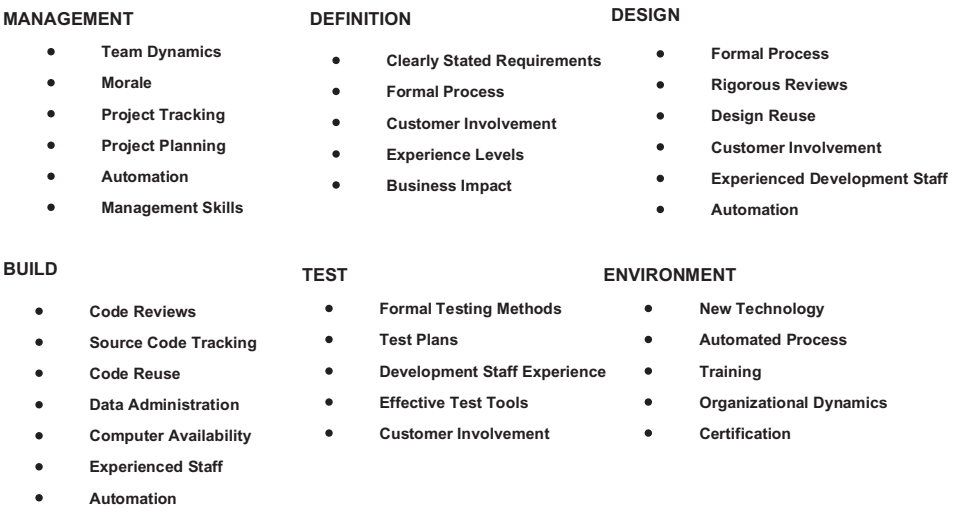


Figure 3: *Factors That Influence Risk*

ware processes utilized, staff skill levels (including user personnel), automation utilized, and the influences of the physical (development conditions) and business environment (competition and regulatory requirements). Categorized in Figure 3 are some examples of influencing factors that must be evaluated to produce an accurate estimate.

As each project commences, the size, complexity, and various risk factors are assessed, and an estimate is derived. Initially, the resulting estimate would typically be based upon industry data that reflect average occurrences of behavior given a project's size, complexity, and performance profile. Over time, as an organization develops a historical baseline of information regarding its own behaviors, performance profiles would reflect a more accurate representation of likely outcomes (see Figure 4). This information can be used to predict and explore what-if scenarios on future projects.

Bringing It All Together

As an example of how the elements of the estimating model fit together, assume we are about to initiate a project that requires an enhancement to an existing system. The first step would be to fully understand the stated requirements and to evaluate those requirements from a functional perspective. Based upon our understanding of the functionality being added, changed, or deleted from the existing system, we would apply the function point method to determine the size of the requirements.

The function point method would consider all new or changed inputs, outputs, inquiries, interfaces, and internal stores of data. Based on a series of weights and algorithms, an unadjusted function point size would be derived. This unadjusted value would then be fine tuned by an examination of the 14 general system characteristics and value adjustment factors noted earlier. The result would equate to an adjusted function point value.

Let us assume in this example that the resulting function point size is 250 function points. This information alone tells us very little about what it will take to make the required changes to the existing system. We need to consider any additional levels of complexity that may inherently be part of the change required, and we need to assess our capacity to design, develop, and deploy a project of this given size.

Industry data indicate that (on average) an enhancement project of a given size would equate to a range of delivery rates. Considering our example's size of 250 function points, we know from standard industry data that we could expect a delivery rate ranging from four to 25 function points per person-month. That is a wide range, which will of course be influenced by a variety of factors, including those previously mentioned.

To effectively complete the estimate (and our example), we must evaluate all of the risk factors that will influence the ability to deliver the required changes. If project data have been collected and analyzed for a statistically relevant period of time, performance profiles would be available that would pinpoint a likely outcome when we matched the current profile of risk factors to an existing profile.

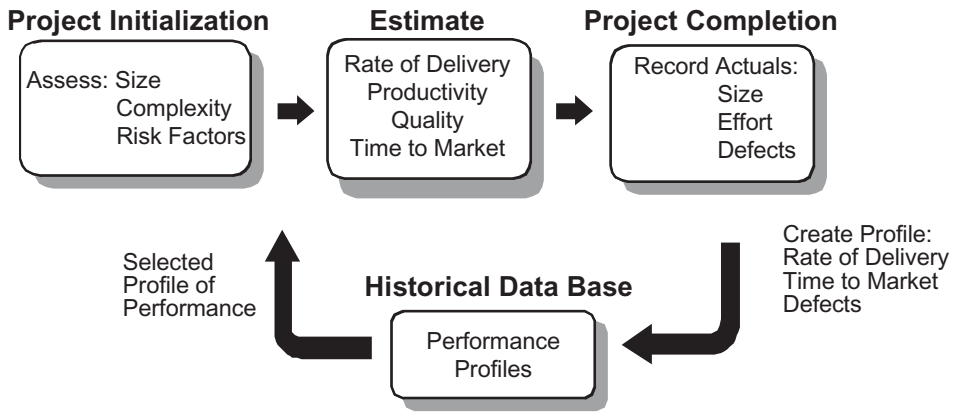
In place of an internal historical base of information, organizations are dependent upon industry data sources. These sources of industry data will produce varied results relative to an organization's actual experiences, but they are of significant value when there is little information available from within.

Industry Data

Companies have not typically invested the resources to develop internal rate-of-delivery performance baselines that can be used to derive estimating templates. Therefore, industry data baselines of performance delivery rates are of significant value. The industry data points allow organizations to use these generic delivery rates to ballpark their estimates. As they continue to develop an experience base of their own, they can transition from using industry data to using their own data.

The International Software Benchmarking Standard Group (ISBSG) is one of several opportunities that currently exist for gathering, retrieving, and sharing industry data. ISBSG operates on the principle of a well-defined collection process that feeds a central repository, making the data available for detailed access and comparison to industry best practices. The advantage of industry databases is the accessibility of detailed data. Information on ISBSG can be found at <www.isbsg.org.au>.

Figure 4: *Developing and Using a Historical Baseline*



Summary

Accurate and early estimating requires the following:

- Proper identification of the problem domain, including functional size and complexity.
- An assessment of the organization's capacity to deliver based upon known risk factors.
- Use of industry data points as necessary to provide delivery rates or as a point of comparison.

As Robert Glass says in *Building Software Quality* [1], "If there is one management danger zone to mark above all others, it is software estimation."

Furthermore, an investment in skills training and risk profile development is critical. Project managers must be equipped with the proper tools and techniques necessary to accurately estimate projects. The return on that investment is obvious to any organization that has misspent dollars because of inaccurate estimating.

Reference

1. Glass, Robert. Building Software Quality. Prentice Hall PTR, 1997.

Additional Reading

1. Garmus, David, and David Herron. Measuring the Software Process: A Practical Guide to Functional Measurement. Prentice Hall, 1996.
2. Garmus, David, and David Herron. Function Point Analysis: Measurement Practices for Successful Software Projects. Addison-Wesley Information Technology Series, 2000.

About the Authors

David Garmus is a principal and founder of The David Consulting Group. He is an acknowledged authority in sizing, measuring, and estimating software application development and maintenance with more than 25 years of experience in managing, developing, and maintaining computer software systems. Concurrently, as a university instructor he taught courses in computer programming, system development, information systems management, data processing, accounting, finance, and banking. Garmus is the immediate past president of the International Function Point Users Group (IFPUG) and a member of the Counting Practices Committee from 1989 through 2001. He previously served IFPUG as chair of the Certification Committee, chair of the New Environments Committee, and on the Board of Directors as director of Applied Programs, vice president, and president. Garmus has a bachelor's of science degree from the University of California at Los Angeles, and a master's degree from Harvard University.

The David Consulting Group
1935 Salt Myrtle Lane
Orange Park, FL 32003
Phone: (904) 269-0211
Fax: (904) 215-0444
E-mail: dcg_dg@compuserve.com

David Herron is a principal and founder of The David Consulting Group. He is an acknowledged authority in using metrics to monitor information technology's (IT) impact on business, including advancement of IT organizations to higher levels on the Software Engineering Institute's Capability Maturity Model® and on the governance of outsourcing arrangements. He assists clients in establishing software measurement, process improvement, and quality programs and to enhance their project management techniques. Herron has more than 25 years experience in managing, developing, and maintaining computer software systems. He serves as a Cutter Consortium Expert Consultant. Herron attended Union College and Northeastern University. He is chair of the International Function Point Users Group (IFPUG) Management Reporting Committee, a member of the IFPUG IT Performance Committee, and a member of the American Society for Quality.

The David Consulting Group
19 Pointe View Drive
Medford, NJ 08055
Phone: (609) 654-6227
Fax: (609) 654-2338
E-mail: dcg_dh@compuserve.com

New Code Analyzes Fluid Flow for Better Designed Aerospace Vehicles and Components

Dr. Greg D. Power
Sverdrup Technology, Inc. AEDC Group

This article describes the Wind Computational Fluid Dynamics code developed by the National Project for Applications-Oriented Research in Computational Fluid Dynamics (NPARC) alliance, a formal partnership between the U.S. Air Force Arnold Engineering Development Center and the NASA Glenn Research Center. The project was nominated to CROSSTALK *in its 2001 search for the top five software projects in the U.S. government. It received high scores by the Software Technology Support Center review team. The NPARC mission was to develop, validate, and support an integrated, general purpose computational flow simulator for the U.S. aerospace community that can be used to analyze fluid flow to better design and develop aerospace vehicles and components.*

The Wind Computational Fluid Dynamics code was developed to analyze fluid flow to better design and develop aerospace vehicles and components. This was the primary software product developed by the National Project for Applications-Oriented Research in Computational Fluid Dynamics (NPARC) alliance, a formal partnership between the U.S. Air Force (USAF) Arnold Engineering Development Center (AEDC) and the NASA Glenn Research Center (GRC).

The NPARC mission was to develop, validate, and support an integrated, general purpose computational flow simulator for the U.S. aerospace community. The NPARC project's success is exemplified by its large number of registered users from government, industry, and academic organizations during the past three years. In addition to AEDC and GRC, other government, commercial, and academic organizations participated in the planning and execution of the project. One of the primary contributors was The Boeing Company.

The Wind code solves the Navier-Stokes equations using a variety of iterative algorithms. The flowfield can be two-dimensional, three-dimensional, or axisymmetric and may represent steady or unsteady flow conditions. Turbulence is simulated using a variety of turbulence models, and the convection, dissipation, and reaction of chemical species can be modeled. A variety of numerical algorithms are available to integrate the flow equations depending on user accuracy requirements. The solution domain may be divided into multiple zones, each of which is transformed into cubicle regions in the computational space.

The code has been applied to a variety of systems, including air-breathing engine inlets and nozzles, liquid and solid propellant rocket propulsion systems, full aircraft systems, missile control systems, test cell and wind tunnel aerodynamics, and hypersonic vehicles. Computational fluid dynamics has become a major contributor to the design and test process for Department of Defense (DoD) weapons systems by allowing the flow around and through systems to be evaluated before costly designs become reality. The result is significant reduction in the overall cost and cycle time of the acquisition process.

Wind Code Specifics

The NPARC alliance consists of diverse organizations across a large geographical area. These organizations perform first-rate operations coordinating configuration management, development efforts, code distribution, and user support. To facilitate this coordination, the NPARC project developed an Internet Version Management System (IVMS) to provide centralized support for multisite, multiplatform software distribution and development.

With IVMS, files are provided to users and developers for each major release. This browser-based system allows developers to log into the system, check out a software module, modify it, and insert it back into the system. IVMS prevents other developers from simultaneously updating the same module. IVMS also allows users to have immediate access to the latest code versions from a protected file transfer protocol (FTP) site.

The Wind code is written in FORTRAN90 and C and includes excellent code reuse examples from the individual efforts of the USAF AEDC, NASA GRC, and The Boeing Company. Excluding libraries maintained by other organizations, the Wind program consists of 160,000 lines of FORTRAN90 code and 18,000 lines of C code. There are also 29 auxiliary programs used for processing. Altogether, these utility programs contain about 240,000 lines of FORTRAN90 code and 75,000 lines of C code.

Documentation is provided on the Wind code and its associated utilities in HTML, PDF, and PostScript formats. The documentation is extremely detailed and constantly updated to reflect changing capabilities and to clarify descriptions of existing capabilities. Developer documentation is also provided on the Web, but is accessible only to registered developers. This documentation consists of information on code structure, memory management, and details on specific modules. During the last year, this Web site has handled, on average, approximately 30,000 user inquiries per month. The documentation and other Web-based information can be accessed at <www.arnold.af.mil/nparc>.

NPARC software is developed primarily in a Unix environment, and it runs on a multitude of platforms, including systems using SGI IRIX, Sun Solaris (Sparc CPUs), Linux (Intel-compatible CPUs), and Hewlett-Packard HP-UX (PA-RISC CPUs). The code has been ported to and used on most major systems at DoD's Major Shared Resource Centers.

Wind Code Application and Operation

The Wind code can run in parallel on a network of workstations or a large multiprocessor machine using either message passing interface (MPI) or parallel virtual machine (PVM) protocols. To run in parallel, the user creates a multi-processor control file containing a list of hosts on which to run. The Wind run-scripts set up an entire virtual machine environment. The scripts check the remote machine to see whether it is overloaded before adding it to the virtual machine. This prevents the job from waiting for the overloaded machine to complete its tasks.

The parallel methodology used is a

master-worker paradigm, where the master controls the work allocation and communications between the workers. The master assigns a grid block to each worker as it completes work on the previous grid block. This approach results in excellent load balancing, provided there are sufficiently more grid blocks than worker processes. Worker-to-worker communication is allowed to improve the overhead for exchanging zonal interface information.

To ensure product quality, a developer's programming approach must be discussed with other developers and approved before implementation. Before modifying any modules through the IVMS, developers must have their work checked by at least one other developer at another organization. A description of the modification and the name of the individual who checked the work must be entered into the IVMS.

To aid in quality assurance, a validation Web site is maintained containing numerous test cases with experimental data for comparison. The Web site contains all of the files and input data required to reconstruct a test case. All output files are also available. When modifications are made to the code, the test cases on this site are used to validate that the modification operates as expected.

A set of standards documents was developed as one of the project's first tasks. A document describing programming guidelines provides a set of required programming practices. A document describing documentation guidelines provides a description of all NPARC documentation and the information required to document new features. Finally, a document describing testing standards provides guidance on procedure, documentation, and achievement of validation and functional testing.

Each year, key customers and the alliance partners hold a NPARC workshop to identify customer and user needs. The NPARC support team is also available year round via a dedicated e-mail address and phone number. User-identified bugs are handled on a priority basis and the modified code is made available to the user community as soon as the bug is fixed.

Project Performance

The NPARC project is an ongoing development effort geared to rapidly respond to customer needs, which require frequent schedule modifications as the environment shifts. Recently, AEDC completed a three-year High Performance Computing (HPC) Modernization Office effort under the Common HPC Scalable Software Initiative (CHSSI). CHSSI milestones, including improved parallelization and usability, were established and were exceeded in most cases. In 1999, the project received an Honorable Mention in the NASA Software of the Year Award competition.

The NPARC project places a huge emphasis on supporting the aerospace community, which is confirmed by the projects' large following. The code has been requested by more than 273 organizations in the past three years. User organizations include 42 DoD organizations, 21 government organizations, 149 commercial companies, and 61 universities. The customers are primarily associated with aerospace design and testing, although there are users associated with other industries. By supporting a large number of users, the United States' principal aerospace organizations (USAF and NASA) benefit greatly from the code's exposure to unforeseen situations as it is applied to configurations and flow conditions that are not normally encountered within a single organization.

This project demonstrates a new way of doing software development in general and governmental software development in particular through the coordinated development efforts between NASA, DoD, and industry organizations using Internet-based tools. The NPARC product is a true government off-the-shelf success story. The unique development environment allows each organization to leverage the high cost of program development with other organizations.

Usually, a development effort of this nature requires investments from $1 to $2 million annually to adequately maintain and support the software tools. By leveraging resources, no single organization must invest more than $500,000 while reaping the benefit of the entire investment. The ultimate beneficiary of these products is the U.S. taxpayer, through prudent cost sharing among government agencies and leveraging resources with commercial and academic partners.

This leveraging also provides the aerospace community with a first-class free product and a greater ability to share data and produce technological advances for the United States. The involvement of commercial organizations in the utilization, planning, and execution of the software development helps to ensure that the United States maintains worldwide economic and military superiority.

More information on all aspects of the NPARC project may be found at <www.arnold.af.mil/nparc>.

Conclusion

The Wind code is a computational fluid dynamics software package used within the government and industry to analyze fluid flow to better design and develop aerospace vehicles and components. This software is jointly developed by the NPARC Alliance, a consortium of partners in the DoD, NASA, and industry, through Internet version management tools and disciplined coordination. While joint development of software is a significant challenge, the NPARC Alliance has demonstrated that a multi-site software development effort can result in a high-quality product and development cost savings.

Additional Readings

1. Slater, J. W., J. C. Dudek, and K. E. Tatum. "The NPARC Alliance Verification and Validation Archive," 2000-FED-11233. Proceedings of ASME FEDSM'00, ASME 2000 Fluids Engineering Division Summer Meeting. Boston, MA, 11-15 June 2000.
2. Matty, J. J., G. D. Power, and W. A. Acosta. "HPC CHSSI CFD-7: Scalable Wind From the NPARC Alliance," AIAA-99-3674, 30th AIAA Plasmadynamics and Lasers Conference. Norfolk, Va, 28 July - 1 June 1999.
3. Bush, R. H., G. D. Power, and C. E. Towne. "WIND: The Production Flow Solver of the NPARC Alliance," AIAA-98-0935, 36th AIAA Aerospace Sciences Meeting and Exhibit. Reno, NV, 12-15 Jan. 1998.

About the Author

Greg D. Power, Ph.D., is a team manager and engineering specialist with Sverdrup Technology, Inc. at the Arnold Engineering Development Center in the Computational Simulation, Modeling, and Analysis group. His responsibilities include the development and application of computational fluid dynamics analyses for use in design of experimental facilities and integrated test and evaluation of aircraft components. He received his doctorate degree from the Georgia Institute of Technology.

Sverdrup Technology, Inc.
AEDC Group
740 Fourth Street
Arnold AFB, TN 37389-6001
Phone: (931) 454-5832
DSN: 340-5832
E-mail: greg.power@arnold.af.mil

Measuring Calculus Integration Formulas Using Function Point Analysis

Nancy Redgate
American Express Risk Management

Dr. Charles Tichenor
Defense Security Cooperation Agency

Function point counters, software developers, and others occasionally need to measure the size and complexity of calculus integration formulas embedded in engineering and scientific applications. Sizing these formulas using function point analysis can result in more accurate measures of application size and improved quality in forecasting costs, schedule, and quality. It can also improve the confidence of those new to the function point methodology as they see that all of their calculus work is recognized and measured. This article shows an approach to sizing these formulas. This methodology is in full compliance with the International Function Point Users Group procedures and does not require any additional counting rules or patches.

Measuring the size and complexity of software is critical to the development of business models that accurately forecast the development cost, duration, and quality of future software applications. One way to measure software size is through function point analysis, especially using the methodology of the International Function Point Users Group (IFPUG) as published in their "Function Point Counting Practices Manual" (CPM) version 4.1 [1]. Readers unfamiliar with function point analysis are referred to the CPM as the primary reference for this methodology.

As good as the IFPUG function point methodology has been, it has historically had a perceived gap regarding the sizing of software algorithms, especially those embedded in real-time software. A proposed solution to the function point community was a general procedure to measure the size of those algorithms, which appeared in CROSSTALK February 2001 [2].

Since publishing that article, the authors were asked to focus this procedure for certain types of algorithms – calculus integration formulas – that can appear in engineering and scientific applications. This article, therefore, focuses the general procedure for sizing algorithms into a specific procedure for sizing calculus integration formulas. This is suitable for experienced function point counters who are also familiar with the fundamentals of calculus.

Description of an Algorithm

An algorithm is a series of equations solved in a logical sequence to produce an external output (EO). Real-time software sometimes contains embedded algorithms. Examples of these can include algorithms for controlling a nuclear reactor, calculating complex pricing agreements, or optimizing production levels in manufacturing. Calculus integration formulas fit this description of an algorithm because, as we will show, their solution requires solving a series of equations in a logical sequence before a solution can be reached.

Every algorithm must have one external input (EI), internal logical file (ILF), and EO with at least the following:

- Data and/or control information must be externally input into the algorithm.

"The authors suggest that ... [those] in mathematical academia or those with strong mathematical skill sets can further this theory."

- Data and/or control information must be logically stored. This storage area is not necessarily shown on an entity relationship diagram or other diagram depicting physical data files.
- The results of the algorithm's execution must be identifiable to the user as an EO.

Integration Formulas as Algorithms

Let us consider the following as an example for using function point analysis to measure integration formula size and complexity.

$$\int_{4}^{10} 6\,dx \qquad (1)$$

The solution of this formula can be expressed as the area under the curve f(x) = 6, as bounded by the x-axis, and within the domain four through 10. Figure 1 shows this graphically as the shaded area.

The Fundamental Theorem of Calculus gives us the algorithm to determine this area. The general formulation of the theorem is as follows.

$$\int_{a}^{b} f(x)\,dx = F(b) - F(a) \qquad (2)$$

In this example, this formulates as follows.

$$\int_{4}^{10} 6\,dx = F(10) - F(4) \qquad (3)$$

This algorithm can be solved graphically. First, calculate the area of the shaded region in Figure 2. This is the region bounded by the points (0,0), (10,0), (10,6), and (0,6). This is F(b), or F(10), and its area is 60 square units.

Then, calculate the area of the striped region in Figure 3. This is the region bounded by the points (0,0), (4,0), (4,6), and (0,6). This is F(a), or F(4) and its area is 24 square units.

Finally, subtract the striped region from the shaded region. This is F(b) - F(a), or F(10) - F(4). The remaining shaded region is shown in Figure 4. Its area is 60 - 24, or 36 square units.

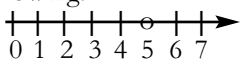
Measuring Size and Complexity

Counting the ILF

Sometimes an ILF is often identified with a physical data table. Here we need to be more specific. According to the IFPUG's CPM, an ILF is "... a user identifiable group of logically related data or control information maintained within the boundary of the application [1]." In mathematical terms, we suggest that an ILF could be described using the follow-

ing set theory: a set of data or control information maintained within the boundary of the application.

A mathematical set can be represented in several ways. For example, we might represent the set of *all numbers greater than five* as the following:

- {all numbers greater than five}, or
- using a number line such as the following:
- 

In this example, therefore, we have an ILF, which is the set of data points in the Cartesian plane required to solve this equation using the Fundamental Theorem of Calculus.

Sometimes ILFs have subgroups of data. These are called record element types (RETs). According to the CPM [1], a RET is a "user recognizable subgroup of data elements within an ILF ..." Using set theory, we could describe an RET as *a subset of data*.

Each ILF is measured by considering the number of data element types (DETs) it logically contains and the number of RETs it contains. A complexity matrix in the CPM then shows how to convert the combination of counted DETs and RETs into function points.

Counting the RETs

The Fundamental Theorem of Calculus states that the set of points (or the area) in the Cartesian plane, which represent the solution to this integration formula, is found by subtracting the area of the striped region of Figure 3 from the area of shaded region of Figure 2. This resulting region has the area shown in Figure 4.

For this example, this entire region has three subsets. The first subset of all points in the Cartesian plane affected by this algorithm is the area of Figure 2, that is, F(b), or F(10). The second subset is the area in Figure 3: F(a), or F(4). The third subset is the area in Figure 4: F(b) - F(a), or F(10) - F(4). This is the solution subset. Therefore, this integration formula has one ILF, with three RETs.

Counting the DETs

The integration formula tells us how to partition the Cartesian plane into the areas F(b) and F(a). Let us recall the formula.

$$\int_{4}^{10} 6\,dx \qquad (4)$$

The ILF is the region of the Cartesian plane bounded by the f(x) on the *north* side, the x-axis on the *south* side, x = 10 on the *east* side, and x = 0 on the *west* side. The solution RET is bounded by x = 4 on the west side. This gives us the information we need to count this formula's DETs.

- The first DET is f(x), the northern boundary of the ILF. In this case, the DET is six.
- The second DET is dx, which tells us that the integration is with respect to x and is therefore bounded in this case by the x-axis on the southern side.
- The third DET is 10, which gives us the "b" in F(b).
- The fourth DET is four, which gives us the "a" in F(a).

Therefore, this ILF has three RETs and four DETs. According to the CPM complexity matrix, this is a low complexity ILF and is worth seven function points.

To generalize this method, we suggest the following key points shown in Figure 5.

- The shaded area on this graph represents an ILF. It is "... a user identifiable group of logically related data or control information maintained within the boundary of the application."
- The RETs are the "blocks" or areas needed to graphically depict F(b), F(a), and F(b) - F(a).
- The number of DETs represents the instances of data and/or control information to define the solution area F(b) - F(a).
- This ILF does not appear in an ER diagram.

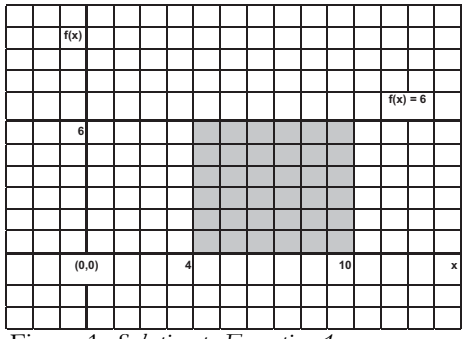


Figure 1: *Solution to Equation 1*

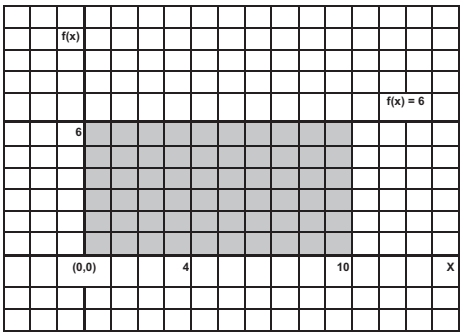


Figure 2: *F(b)*

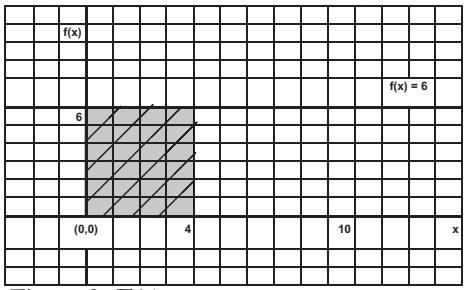


Figure 3: *F(a)*

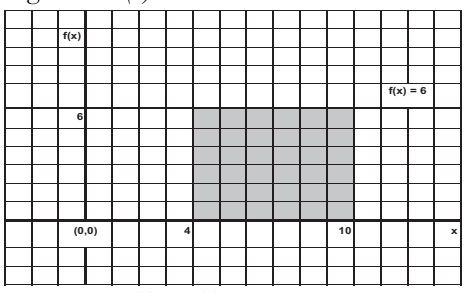


Figure 4: *F(b) - F(a)*

Counting the EI

One can imagine a simple version of an input screen having a layout as follows:

f(x) = ☐
dx or dy ☐
b = ☐
a = ☐
ENTER

There are four DETs as stated, plus the control information ENTER key to affect the input process. There is one ILF being referenced (File Types Referenced [FTR]), which is the graph of Figure 5. The CPM complexity matrix shows that an EI with four DETs and one ILF is a Low complexity EI worth three function points.

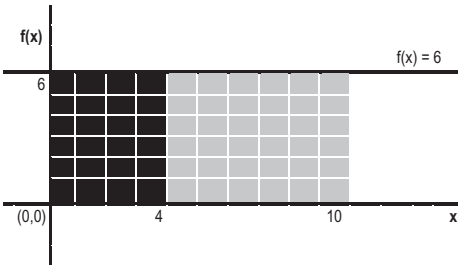
Counting the EO

Imagine a simple report with the solution of the integration formula calculations – perhaps a paper report that looks like this.

The solution is ☐

In this case, there is one DET on the report, and one FTR – the graph of Figure 5. Therefore, the CPM complexity matrix is used to show that this is a Low EO worth four function points.

Figure 5: *Equation 1's Internal Logic File*



Coming Events

July 18-20
Shareware Industry Conference
St. Louis, MO
www.sic.org

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

July 22-26
6th Annual PSM Users' Group Conference
Keystone, CO
wwwpsmsc.com

August 19-22
The 2nd Software Product Line Conference
San Diego, CA
www.sei.cmu.edu/SPLC2/

September 9-13
International Conference on Practical Software Quality Techniques (PSQT) 2002 North and International Conference on Practical Software Testing Techniques (PSTT) 2002 North
St. Paul, MN
www.psqtconference.com

November 18-21
International Conference on Software Process Improvement
Washington, DC
www.software-process-institute.com

April 28-May 1, 2003
Software Technology Conference 2003

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

Other Examples and Their Solutions

Consider taking a function point count of the following integration formula.

$$\int_4^{10} x^2\,dx \qquad (5)$$

In principle, this formula's ILF is counted the same way as the previous example. There are four DETs (x^2, dx, 10, and 4). There are three RETs in the ILF (F(10), F(4), and F(10) - F(4)). This is a low complexity ILF, worth seven function points. We assume that the EI input screen and EO are similar to the first example, so the total unadjusted function point count is 14.

Consider a slightly different formula.

$$\int_4^{10} (x^2 - x)\,dx \qquad (6)$$

We count the function points using the same logic.

The solution to this formula requires us to break this into two parts. We first use the Fundamental Theorem of Calculus to find the following.

$$\int_4^{10} (x^2)\,dx \qquad (7)$$

Then we subtract from it the solution to the second part.

$$\int_4^{10} (x)\,dx \qquad (8)$$

This ILF therefore contains six RETs – three regions from the first part and three regions from the second part. There are four DETs needed to solve the first part (x^2, dx, 10, and 4) and four to solve the second part (x, dx, 10, and 4). Therefore, this ILF of six RETs and eight DETs is an average complexity ILF worth 10 function points.

One point needs to be clarified here for the advanced function point counter. Even though the dx, 10, and four might appear to be counted twice in this ILF, this does not violate the CPM rule that each DET must be unique. They are actually unique here because the Fundamental Theorem of Calculus must be executed twice – once for each part. Each instance of dx, 10, and four must be used in order to solve the formula.

Finally, consider this formula.

$$\int_4^{10} 3x\,dx \qquad (9)$$

The solution to this requires first simplifying to the following.

$$3\int_4^{10} x\,dx \qquad (10)$$

This formula's ILF still has its three RETs, but its DET count increases from four to five. This is because the solution area of the following is multiplied by three.

$$\int_4^{10} x\,dx \qquad (11)$$

Further Notes for Function Point Counters

The above examples were intended to be simplistic to illustrate the procedure. In more complex cases, we might want to consider whether, for example, there is an add, change, and delete capability associated with the EI. If so, count up to three EIs accordingly (one for the add capability, one for the change capability, and one for the delete capability.) The EO may be more complex, or several EOs might be required. There might be an EQ capability to view what data and control information is currently in the ILF. Finally, an external interface file could be involved.

If an integration formula can be formulated in several ways, extend the concept of the elementary process and choose the smallest unit of activity meaningful to the user. For example, perhaps choose the ILF formulation having the smallest number of RETs.

Algorithms may influence the general systems characteristics. You may need to check, for example, the following:

- GSC5 Online Data Entry.
- GSC8 Online Update.
- GSC9 Complex Processing.
- GSC14 Facilitate Change.

Areas for Future Research

The authors suggest that readers in mathematical academia or those with strong mathematical skill sets can further this theory. The authors suggest that a variety of integration techniques can be counted and that research should be conducted to expand this methodology.

Conclusion

Function point analysis can be used to measure the size and complexity of algorithms in general, and integration formulas in particular. Advanced function point counters need to recognize all algorithms in the software they count, to include

those embedded integration formulas. The IFPUG function point methodology can be used to measure the size and complexity of these integration formulas without the need for additional patches or counting rules.

Reference

1. International Function Point Users Group. Function Point Counting Practices Manual Release 4.1, 1999. Refer to <www.ifpug.org/publications/manual.htm>.
2. Redgate, Nancy, and Charles B. Tichenor. "Measure Size, Complexity of Algorithms Using Function Points." CROSSTALK Feb. 2001: 12-15.

Additional Reading

1. International Function Point Users Group. "Function Points as Assets." (Provides direction for using function point analysis to develop cost, schedule, and quality forecasts.)
2. Garmus, David, and David Herron. Measuring the Software Process: A Practical Guide to Functional Measurements. Upper Saddle River, N.J.: Prentice Hall PTR, 1996.
3. Jones, Capers. Applied Software Measurement: Assuring Productivity and Quality. New York: McGraw-Hill, 1991.

About the Authors

Nancy Redgate has a bachelor's degree in industrial engineering/operations research from the University of Massachusetts at Amherst. She received master's degrees in operations research, statistics, and business administration from Rensselaer Polytechnic Institute.

5 Wood Hallow Drive
34-02-01
Parsippany, NJ 07054
Phone: (973) 526-6602
Fax: (973) 526-3635
E-mail: nancy.redgate@prodigy.net

Charles B. Tichenor, Ph.D., serves as an information technology operations research analyst for the Department of Defense, Defense Security Cooperation Agency. Dr. Tichenor holds a part-time position as an adjunct faculty member at Strayer University's Anne Arundel, Md. campus. He has a bachelor's degree in business administration from Ohio State University, a master's degree in business administration from Virginia Polytechnic and State University, and a doctorate degree in business from Berne University.

Defense Security Cooperation Agency
1111 Jefferson Davis Hwy., East Tower, Suite 303
Arlington, Va. 22202-4306
Phone: (703) 601-3746
Fax: (703) 602-7836
E-mail: tichenor@erols.com

WEB SITES

Software Cost Estimation Web Site

www.ecfc.u-net.com/cost/index.htm

The Software Cost Estimation Web site presents a review of current cost estimation techniques to help industry and academia choose the appropriate methods when preparing software cost estimates. The site covers both traditional and state-of-the-art methods identifying advantages and disadvantages of each and the underlying aspects in preparing cost estimates. The site also provides links to other software cost estimation sites that are involved in this area and details the research that has been undertaken at Bournemouth University.

International Function Point Users Group

www.ifpug.org

The International Function Point Users' Group (IFPUG) is a non-profit organization committed to increasing the effectiveness of its members' information technology environments through the application of function point analysis (FPA) and other software measurement techniques. IFPUG endorses FPA as its standard methodology for software sizing and maintains the "Function Point Counting Practices Manual," the recognized industry standard for FPA.

Practical Software and Systems Measurement Support Center

www.psmsc.com

The Practical Software and Systems Measurement (PSM) Support Center is sponsored by the Department of Defense (DoD) and the U.S. Army. It provides project managers with the objective information needed to successfully meet cost, schedule, and technical objectives on programs. PSM is based on actual measurement experience with DoD, government, and industry programs. The Web site also has the most current version of the PSM Guidebook.

The Software Productivity Consortium

www.software.org/default.asp

The Software Productivity Consortium is a nonprofit partnership of industry, government, and academia. It develops processes, methods, tools, and supporting services to help members and affiliates build high-quality, component-based systems, and continuously advance their systems and software engineering maturity pursuant to the guidelines of all of the major process and quality frameworks. Its Technical Program builds on current best practices and information technologies to create project-ready processes, methods, training, tools, and supporting services for systems and software development.

Software Estimation: Perfect Practice Makes Perfect

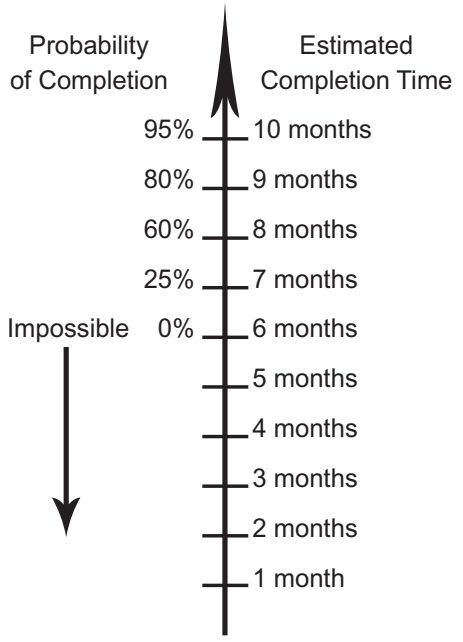
David Henry
Linux NetworX

Accurate software estimation has long been a headache for software developers. Much of the problem stems from lack of estimation training and practice. This article is the result of the author's in-the-trenches experimentation with different estimation training methods and is intended to give practical advice to managers or technical leads who wish to initiate a software estimation process in their organization

Software estimation is a difficult problem. Some have called it a *black art*. How does one go about learning this black art? Actually, it is not as exciting as it sounds: It takes practice and hard work. Unless developers get regular practice doing estimation work, improvement will be difficult. My son's karate instructor shouts his philosophy about practice at the end of each karate session: "Practice doesn't make perfect! Perfect practice makes perfect!"

Most software developers do some estimation work, but many are not trained to do it properly. Regular estimation work with feedback gives the developer the opportunity to improve his/her estimation skills. This article presents a few ideas about how to involve software developers in the estimation process. The techniques presented are being used at the author's organization with some success; result data will be presented later in the article.

Figure 1: *Estimates Are Probability Statements*



Why Software Projects Are Poorly Estimated

There are many misconceptions among software developers about software estimation, which leads them to create poor estimates. The first thing a software developer should understand is *what* an estimate really is; estimates are probability statements. For example, if a developer believes that he/she has an 80 percent chance of completing a project in nine months (see Figure 1), but his/her manager says it has to be done in seven months, what just happened?

By moving the completion date arbitrarily, the manager has just reduced the probability of on-time completion to 25 percent (assuming no other parameters are changed). This was probably not the manager's intention. Most people would probably feel very uncomfortable if they knew the project they were responsible for only had a 25 percent success probability. They would most likely make a vigorous effort to move the completion date back. By understanding and teaching the concept of estimates as probabilities, engineers can make a better defense of their estimates.

Software engineers should also understand the difference between *target setting* and *estimation*. Target setting is when a completion target is set because of some external dependency, such as a conference or fiscal year, and the engineer has to figure out how to meet that target. True estimation must be based on an analytical foundation with an estimation process that is not open to debate. The estimation process should be a black box with inputs of requirements, resources, etc., which generate the estimate (see Figure 2).

Inputs are *independent* variables and the output is a *dependent* variable. For example, if the manager wants to shorten the schedule, he/she must experiment with different inputs: adding more resources to the project or reducing the functionality of the end product. A defined estimation process is critical to any organization that desires repeatable, consistent, and quality estimations.

Another common problem is that the estimation task itself is often not scheduled as part of the project. This leads to what is known as *off-the-cuff* or *best-guess* estimates (or euphemistically as *expert judgment*) that most developers at one time have done and later regretted. It can be difficult to justify the importance of estimation work when there is a lot of pressure to start working on project deliverables.

The estimation work at the outset of a project is really just the tip of the iceberg: Creating infrastructure and the culture to collect and analyze metrics and other estimation inputs throughout the project can take a lot of organizational discipline and work. It is easier to use personal memory of past projects than to gather and analyze historical data. However, creating estimates from personal memory alone is a proven cause of cost and schedule overruns [1].

Is a high level of estimation accuracy even possible early in the project? The answer is, "It depends" (more about this later). In a less mature software organization, there are usually too many unknowns at the beginning of a software project to estimate the following with high precision and accuracy: Which technology will be used? How long will it take to learn it? What if it doesn't work as advertised? The list goes on and on. The author has been involved in many projects where project milestones are all mapped out at the beginning of the project (before critical decisions have been made) with pinpoint precision. This false precision gives the impression that the estimates are also accurate.[1]

With so many unknowns, estimating software can be like peering into thick fog. One can see things that are near pretty clearly and estimate their distance.

Objects farther away can only be dimly seen or not seen at all. How accurate can distance be estimated when the traveler has no map and cannot see the goal, but is relying on memory only? Early estimates should be presented in a way that expresses this potential inaccuracy. For example, the author has attended meetings where managers presented all completion estimates to the day (even long-term estimates). Once when one of these estimates was being presented (which was still over a year away), the question was asked, "Do you want the project completed at 10 a.m. or 11 a.m.?" That manager got the message and now gives all estimates farther than three months away in quarter precision, e.g., first quarter of 2002. Treating early estimates for what they are takes a lot of unnecessary pressure off the developers. Of course, estimates can and should be revised as the project progresses, since the end can usually be more clearly seen the closer it gets.

Now to explain the answer given earlier, "It depends." There are some software projects that are estimated with high precision *and* accuracy. If an organization has a tuned estimation process, metrics gathering, risk management, and other necessary processes in place, highly accurate estimates are certainly possible. The fog of project unknowns will bother this type of organization less. Its trained engineers have created an organization-specific map of the software development terrain and can more easily navigate around roadblocks. There are other situations where high accuracy can be obtained even with a less mature organization: projects where the engineering team has high familiarity with the domain and small non-complex projects.

Where to Start

Before proceeding to educate developers about software estimation, it is instructive to find out how skilled at estimation they are. Some people have a knack for the basic skill of estimating, and others need more practice. Try taking Jon Bentley's estimation quiz [2].[2] When taking the quiz, instruct the engineers to fill in upper and lower bounds that give one an 80 percent chance of including the correct value. If the engineer scores poorly, this is a chance to remind him/her to read requirements more carefully. The whole point of taking an estimation quiz before getting down to the nuts and bolts of estimating is to establish the *before* state, so the engineer can track his/her improvement at basic estimation skills.

The logical place for the engineer to do estimation work is within the scope of the organization's estimation process. If no estimation process exists, get one in place first. Defining the estimation process is outside the scope of this paper, but some online resources are given later.

Create a Feedback Loop

Developers need to keep track of their own day-to-day estimates to estimate accurately in the small. Estimation in the small is a different problem than estimation in the large, but it is an easier problem that should be tackled first. Most engineers will not need to do much estimating in the large unless they are technical leads or managers, so this paper will not address its unique problems.

Recently we interviewed a candidate for a software development position, and asked him how he liked to be managed. He replied: "I write embedded code for devices. Here's how it works – a sensor reads a device, providing input to the program, which then modifies the output to the device. The sensor repeatedly gets new values from the device, which allows the program to guide the device until the sensor detects values in the correct range. This means that the device is operating at the proper level."

He was describing a feedback loop, which is how he liked to be managed (with the manager as the program and the employee as the device). When managing people, a self-directed feedback loop is even better, which mostly takes the manager out of the picture. We have implemented such an e*mbedded controller* in our organization.

We have created an estimation spreadsheet that contains estimates for tasks in the product requirements. (The spreadsheet is available at <www.burgoyne.com/~henryd/estimation>.) At the beginning of each week, the engineer enters estimates for the weeks' work and the actual amount of time the task from the previous week had taken. This is the essential idea, and it only takes a few minutes per week. The spreadsheets are checked into our source code repository, so anyone can see others' spreadsheets.

The spreadsheet contains some simple formulas to allow the engineer to see at a glance how close his/her estimates are to reality. Since it is a spreadsheet, it can also be easily modified to suit different needs, and charts or graphs could be added. The manager can see who is currently working on what and how well the engineer is doing at estimation.

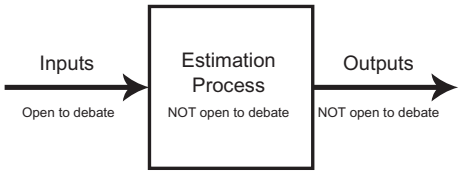


Figure 2: *Estimation Process as a Black Box*

There are certainly other ways of keeping track of what work is currently being done, but this solution is lightweight and does not require expensive or custom software. The main point is that the engineer should be teaching himself to estimate by practicing estimation on a regular basis and seeing the accuracy and results of those estimates.

There are a few caveats, however. As with any metrics gathering, it is best to explain at the outset that the gathered information will not be used *against* the employee (e.g., in reviews). Otherwise, engineers may fudge the numbers, rendering the data useless. We have found it useful to only enter time spent on the actual task, not including interruptions, personal time, etc. Because of this, the estimation spreadsheet does not differentiate between *actual* and *elapsed* task time. Elapsed time is the time between the start and finish points of a task (including interruptions and time spent on other tasks), whereas actual time is the time spent only on the task. We felt that elapsed time was more of a scheduling issue and did not belong in a tool used only for estimation purposes, although other organizations may see this issue differently. In any case, standardizing the metric helps simplify the process.

The tasks themselves should be small tasks of three days or less in size. This will force the developer to think about the details of the task, and the estimation should therefore be more accurate when broken into smaller pieces. For example, if a large task estimate (30 days) is underestimated by 50 percent, the estimation error will be larger than if the same task is broken into 10 smaller three-day tasks. It is unlikely that all 10 tasks will be underestimated by 50 percent: some will be underestimated and some overestimated, which will tend to cancel the estimation errors and produce a better overall estimate.

The developer should give three estimates: worst case (pessimistic), best case (optimistic), and most likely. When asked to give single-point estimates, many developers will simply provide best-case

estimates. Giving all three estimates will reveal such tendencies.

Our estimation spreadsheet has been in use for approximately eight months at the author's organization by engineers with little or no previous formal estimation practice. Before attempting any work-related estimation, all engineers involved in the project took the estimation quiz mentioned previously and scored poorly. During the first month of use, the average percent differences between estimates and actual task completion times were about 75 percent, and the variance in the group was quite large: 25 percent to 150 percent. By the third month of use, the average percent differences for the month had dropped to about 35 percent, with the greatest improvements among engineers who had never before given estimates in such detail. The average differences had stabilized to about 25 percent after six months, which may be the best average results to be obtained by this method.

Comments from the engineers suggest that the most useful feature of this method is the estimation history contained in the spreadsheet, which assists the engineer's memory when creating new estimates. Other comments suggest that this process helped the participants properly size lower- and upper-estimate bounds (shown as best- and worst-case estimates in the spreadsheet). This lightweight method seems to work best as an introduction to software estimation. If the organization wishes or needs better estimation accuracy, this method can be modified or discarded for a more rigorous method once the software team gains some estimating skill.

Use Group Estimation Processes

When making project estimates, get everyone's head into the game by using group estimation techniques. Wide band Delphi is a popular formal technique [3]. For those who prefer less formal methods, estimates can be made separately by members of the group and then averaged. This tends to have a conservative effect on the estimate. The wide band Delphi technique is good for groups with strong personalities because of its anonymous estimations. If strong personalities are not a problem, try reaching group consensus, but do not vote to overrule someone who may have a valid point.

We use another spreadsheet to aid our group estimation process. By first decomposing our project by requirement into modules/classes, each engineer separately estimates the size of each module in lines of code (LOC). In order to estimate size, we use estimation by analogy. For example, on a past project five new forms were to be added to the graphical user interface. By counting the LOC per form from the most recent project, it was fairly easy to compare new forms with previously built forms and estimate how many LOC the new forms would contain. After estimating separately we met, and our estimates were surprisingly similar. We discussed differences, which were mostly different assumptions about the requirements. We found this an effective way to create consensus that leads to a sense of *estimation buy-in* among the team.

Lessons Learned

The following lessons learned are suggested for best results:

- Do not rush estimates. Take the time to define and *use* a formal estimation process.
- Integrate regular estimation work into organizational processes.
- Use group estimation techniques, allowing team members to learn estimation techniques from one another.
- Gather estimation measurements and use them in future estimation efforts.
- Avoid *off-the-cuff* estimates. Managers should not be tempted to demand these types of estimates. Developers should resist giving an estimate until a detailed analysis of the problem has been made.

References

1. Lederer, A., and J. Prasad. "Nine Management Guidelines for Better Cost Estimation." Communications of the ACM Feb. 1992: 34-49.
2. Bently, Jon. Programming Pearls 2nd Ed., appendix 2. Addison-Wesley, 2000.
3. Wiegers, Carl. "Stop Promising Miracles." Software Development Feb. 2000.

Notes

1. For a good discussion of the difference between accuracy and precision, see Steve McConnell's "Rapid Development," Microsoft Press, 1996: 173.
2. The estimation quiz is available online at <www.cs.bell-labs.com/cm/cs/pearls/quiz.html>.

Free Estimation Software

- Construx Estimate 2.0. A user-friendly tool that combines several estimation methods. <www.construx.com/estimate>.
- COCOMO. Requires knowledge of COCOMO II. <http://sunset.usc.edu/research/cocomoii>.
- Cosmos. Requires knowledge of COCOMO II. <www-cs.etsu.edu/softeng>.
- SizeCost. A wizard-oriented tool for the beginner estimator. <http://members.tripod.com/~djelovic/sizecost.htm>.
- SoftEst. A full-featured COCOMO II implementation. <http://sepo.sparwar.navy.mil/estimation.htm>.

Author's Note

The author is not affiliated with any of the estimation tools noted above. There are of course many excellent estimation tools that can be purchased, but for an organization/developer just starting to work with estimation software, it may be easier to check out some of the simpler free tools first.

The estimation process should be tailored to fit the organization. There are some excellent estimation processes available publicly, including "Manager's Handbook for Software Development," Revision 1 (See section 3: Cost Estimating, Scheduling, and Staffing). It is available online at <http://sel.gsfc.nasa.gov/website/documents/docs/84-101.pdf>.

About the Author

David Henry is director of Software Engineering at Linux NetworX, a cluster computing company. He is also a principal of Synergy Software, a consulting firm, and occasionally teaches programming classes at Columbia College in Salt Lake. Henry has nine years of industry experience in software development. He has a bachelor's degree in computer science from Brigham Young University and is completing a master's degree in computer science at Colorado State University.

Linux NetworX
8689 South 700 West
Sandy, UT 84070
Phone: (801) 562-1010
Fax: (801) 568-1010
E-mail: dhenry@lnxi.com

My Fair Estimate

The room is full of tension. White boards are plastered with convoluted notes etched in multi-colored dry erase ink. Walls are awash with diagrams (affinity, fishbone, entity relationship, and state), charts (flow, Pareto, PERT, and GANTT), structures (breakdown, data, and control), and lists (personnel, resource, and equipment). Coffee is cold, tempers hot, discussion long, forbearance short, donuts fresh, and ideas stale.

This quotidian scene subsists in software war rooms far and wide. At the advent of a new customer, project, or requirement, managers marshal troops to answer two very elusive questions: how long and how much? These simple questions set in motion conjecture, machination, negotiation, and arm wrestling that would nauseate Johnny Cochran. For all our vaunted powers of ratiocination, software engineers tend to be a fickle lot when it comes to estimation. Why?

Being masters of our domain and desiring to be worthy of the vaunted title of engineer, we ignore the fact that our estimates are inherently subjective. For the past decade, software's leitmotif is that software development is analogous to industrial manufacturing. The analogy hints that software construction can be shaped into a repeatable process where programmers are cogs in a Fredrick Taylor production line. While similarities exist in some areas of software development, estimation is not one. The theory cloaks the software estimation process with a farrago of formal notation and hints at objectivity.

In manufacturing, repeatable and codified processes lead to objective measures and estimates. Software development, on the other hand, is an intrinsically creative activity that differs each time code is manufactured. What you composed on your last project rarely translates objectively to your subsequent project. It resembles Bob Fosse's chorus line more than Fredrick Taylor's production line.

Before the maturity pundits kvetch like contumacious sports stars to impugn my opinion, let me explain. I concur that mature organizations are using repeatable processes, but I contend that the complexity of each project varies. In developing software for the F-16 Head-Up-Display, I used the same process and techniques to construct the "Altitude Low Warning" module and the "Enhanced Envelop Gun Site" module. Yet the complexities involved in constructing those two components were about as close as Bill Gates and Larry Ellison. Bollinger elaborates this point in his *IEEE Computer* article "The Interplay of Art and Science."[1]

Variation in complexity, which is difficult to objectively measure, dominates a software project estimate. Consequently, estimating software is unavoidably subjective. Therefore, as we estimate our projects, instead of emerging as the professor of estimation we end up more like Gilligan.

Second, we prefer precision to accuracy. Software engineers favor specific single-value estimates, which are certain to be wrong, over a range of values that have a high probability of enclosing the correct estimate. This concept should not be foreign; we use it all the time. When a spouse asks what time we will be home, we always give a range because we know that if we answer 4:12 p.m. and waltz in at 4:15 p.m., we are sleeping on the couch.

Then there is the Pygmalion effect? From Greek mythology, Pygmalion was a king of Cyprus who carved and then fell in love with a statue of a woman. Psychologists Robert Rosenthal and Leonore Jacobson attached Pygmalion's name to the observation that when evaluating something, the evaluator is hardly neutral, and the evaluator's expectations influence the evaluation.

This was personified eloquently in Bernard Shaw's play "Pygmalion" in which phonetics professor Henry Higgins tutors a Cockney flower girl, Eliza Doolittle, in the refinement of speech and manners. For those who avoid the theatre you may have caught the story in the musical "My Fair Lady?" If you are still not with me, join the theatrically impaired and visualize "Pretty Woman" with interesting dialogue and wit.

In this yarn, the project at hand is the transformation of Eliza into a lady. Participants in this transformation are: Professor Higgins who, despite his love of Eliza, can never truly commit himself fully; Freddy Hill who is naively infatuated with Eliza; and Colonel Pickering who seems aloof of the antics but always seems to be there at the right time with the right words.

How does this apply to estimation? Stakeholders are about as focused on estimation accuracy as my son is on picking up after himself. They provide specious estimates to impress clients and, like my son, are improvident to the mess they leave behind. Stakeholders are more callow than a freshman engineer at a fraternity party. They, like young Freddy Hill, are in love with a project's prospects with little concern for the consequence of their credulous desires.

Software engineers, like Professor Higgins, are more than willing to demonstrate their knowledge, wisdom, and prowess but are short of committing to the minutiae of the project's long haul. We are prone to embellish the estimate to assure that our reputation, health, and marriage remain in tact.

Estimations involving human intervention are prone to the Pygmalion effect, and software estimation is no exception. Exuberant stakeholder expectations counter engineers who, if the truth were known, fervently wish they could get home from the office earlier and come in less on weekends. The fact is no one is a good (impartial) judge of one's capability because our perception of the problem causes bias.

That's where Colonel Pickering comes in – a prudent counselor who mixes analysis with common sense. A sage that applies experience, intuition, and judgment to obviate subjectivity, employ flexibility, and temper bias. A team needs a Pickering to mediate between stakeholders and engineers and swing back the estimation pendulum from fallacious exactitude to viable accuracy.

Higgins forewarns, "… you can come back or go to the devil: which you please."

— Gary Petersen,
Shim Enterprise, Inc.

1. T. Bollinger. "The Interplay of Art and Science in Software ." *IEEE Computer* Oct. 1997.

Dynamic Workshop Set in the Playground of the West

The Air Force's Software Technology Support Center (STSC) is sponsoring **Software Best Practices: An Executive's Perspective**, part of the 2002 workshop series to be held July 16-17. This informative workshop is **FREE** to U.S. government employees and will be held in the vicinity of Hill Air Force Base and the majestic Wasatch Mountains.

This workshop is designed for executives and other high-level managers who are responsible for software acquisition, development, or maintenance activities. Any executive with direct responsibility for software-intensive systems or who works with software-development organizations will greatly benefit from attending.

Attendees will walk away with the ability to:

- Understand the "big picture" of Systems Engineering
- Relate senior management's role to key life-cycle activities
- Understand critical issues and measurements
- Recognize the value of Best Practices
- Know what to ask for during Program Reviews
- Know your role and responsibilities in Software Process Improvement initiatives

Be sure to catch the two remaining workshops in the series, **Object-Oriented Software Development: The Basics for Project Managers and Practitioners** on August 13-15 and **Life-Cycle Software Project Management** on September 17-19. These workshops will fill up quickly, and seating is limited. **Act quickly.**

Photos courtesy Utah Travel Council/Frank Jensen

For additional information visit our Web site at **www.stsc.hill.af.mil**.

SPACE IS LIMITED. To reserve your place at any of these workshops, contact Debra Ascuena at 801-775-5778 (DSN 775-5778) or debra.ascuena@hill.af.mil.

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

Published by the Software Technology Support Center (STSC)

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

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