Software Estimating Technology: A Survey

Richard D. Stutzke, Science Applications International Corp.

http://www.stsc.hill.af.mil/crosstalk/1996/05/Estimati.asp

Introduction

Our world increasingly relies on software. There is a seemingly insatiable demand for more functionality, interfaces that are easier to use, faster response, and fewer defects. Developers must strive to achieve these objectives while simultaneously reducing development costs and cycle time. Above all, senior management wants software delivered on schedule and within cost, a rarity in past software development projects. Software process improvement (SPI), as advocated by the Software Engineering Institute (SEI), helps to achieve these objectives. Project planning and tracking are identified as two key process areas in the SEI's Capability Maturity Model.

Software cost and schedule estimation supports the planning and tracking of software projects. Estimation is receiving renewed attention during the 1990s decade to cope with new ways to build software and provide more accurate and dependable estimates of costs and schedules. This article discusses problems encountered in software estimation, surveys previous work in the field, describes current work, provides an approach to improve software estimation, and offers some predictions and advice concerning software estimation.

Description of the Problem

The estimator must estimate the effort (person hours) and duration (calendar days) for the project to enable managers to assess important quantities such as product costs, return on investment, time to market, and quality. The estimation process is difficult for several reasons. The first problem is that projects often must satisfy conflicting goals. Projects to develop (or maintain) software must provide specified functionality within specified performance criteria, within a specified cost and schedule, and with some desired level of quality (absence of defects). Software engineering processes can be chosen to meet any one of these project goals. Usually, however, more than one goal must be satisfied by a particular project. These multiple constraints complicate the estimation process.

The second problem is that estimates are required before the product is well defined. Software functionality is difficult to define, especially in the early stages of a project. The basis for the first good cost estimate is usually not available for totally new systems until the top level design has been defined. (This design is an instance of the product's "software architecture.") This level of design is only defined at the preliminary design review in Department of Defense contracts (and sometimes not even then, which leads to undesirable consequences). This milestone is reached after about 20 percent of the total effort and 40 percent of the total duration have been expended by the project staff. At this point in a project, typical accuracies for the estimated effort and duration are within 25 percent of the final project actuals. In general, since more information becomes available, e.g., product structure and size and team productivity, the accuracy of estimates increases as a project proceeds. Figure 1 illustrates this and is adapted from Barry W. Boehm [1]. (Commercial software projects behave similarly.) To reduce costs, as well as to improve quality and reduce development times, some projects employ pre-defined "domain specific software architectures" (DSSAs). The development costs for such projects can be estimated more accurately than for projects that build a totally new product since more information about the product is available earlier. (Advanced Research Projects Agency and the SEI, among others, are sponsoring work on DSSAs.) In general, however, the estimator must apply considerable skill to estimate project cost and schedule early in a project.

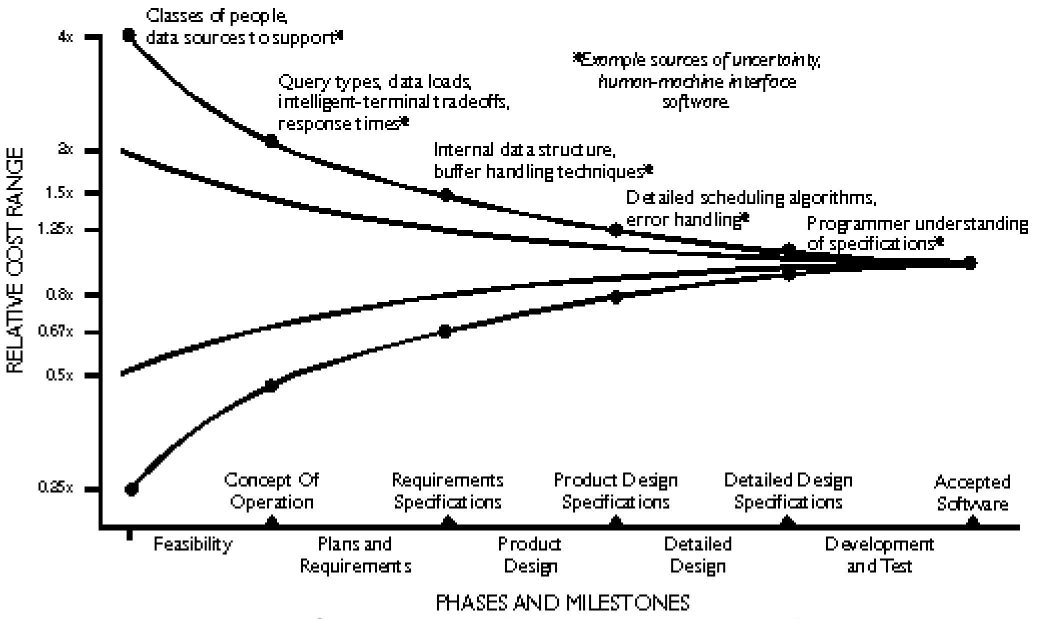

Figure 1: Software Cost Estimation Accuracy vs. Phase.3

For modifications of existing code, more data is available earlier, so more accurate estimates are possible for this kind of project compared to totally new development. This is important since about a half of all software maintenance work is a response to changes in the original requirements or in the system's external environment (mission, interfaces to other systems, etc.) and involve modification of existing code. Software processes must produce software that can be gracefully evolved at reasonable costs. The choice of the software architecture significantly influences modifiability and hence maintainability. (Architecture based reuse is another motivation for the work on DSSAs.) New ways to estimate the costs of such projects are being developed.

Modification of code is closely tied to software reuse. Various cost estimation models have been developed to qualify the economic costs of building and using reusable components. For example, Richard Selby [2] analyzed costs at NASA's Software Engineering Laboratory and found that there is a large increase in programmer effort as soon as a programmer has to "look inside" the component. The cost to reuse a component depends on its suitability for the intended application, its structure, and other factors. Figure 2 illustrates this effect by showing the cost to reuse software compared to the cost to develop new software as a function of the amount of existing code that must be modified to install the code in a new system. For example, modifications may be necessary to accommodate a new operating environment (such as porting from Berkeley Unix to DEC Ultrix). Figure 2 is based on a model I developed using data from Richard Selby [2], Gerlich Rainer and Ulrich Denskat [3], and Barry Boehm, et al. [4]. The two curves shown correspond to the best and worst cases for reuse based on the factors mentioned above. There are several noteworthy features of these curves. First, the cost is not zero even if no code is modified. Second, the costs increase faster than linearly at first. Third, the cost to modify all of the existing code is more expensive than to develop the code from scratch. (Effort is wasted to understand, then discard the existing code before the new code is written. This effort is never expended if the decision is made to develop totally new code from the start.) As shown in the figure, for the worst case, the economic breakeven point occurs when only 20 percent of the code is modified; reuse is not cost effective above the breakeven point. Such nonlinear behavior is not handled in existing cost models.

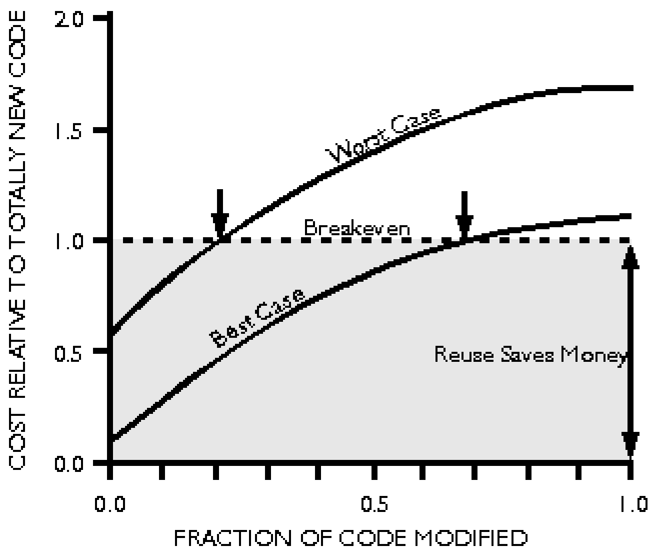

Figure 2: Reusing Software Is Not Always Cost-Effective.

A third complication arises because the way software is being built is changing. New development processes emerge and new ways are needed to estimate the costs and schedules for the new processes. These processes endeavor to provide higher quality software, i.e., fewer defects, produce more modular and maintainable software, and deliver software (products and prototypes) to the end user faster. To meet these and the other objectives (stated previously), developers use combinations of pre-built code components and labor-saving tools. For example, much programming is being put into the hands of the users by providing macro definition capabilities in many products. These capabilities allow users to define sequences of frequently used commands.1 A slightly more sophisticated approach is to allow domain experts to construct applications using special tools such as fourth-generation languages and application composition tools. Larger systems intended for specialized (one of a kind) applications are often built using commercialofftheshelf products to provide functionality in areas that are understood well enough to be standardized. Examples are graphical user interfaces and relational database management systems. The trend toward object oriented languages and Ada supports this by making it easier to develop "plug compatible" components. Thus, reuse of code becomes an increasingly large factor in cost estimation, and a good understanding of the cost factors associated with software reuse becomes even more important.

Lastly, some authors like R. Selby [5] are advocates of "measurement driven development processes" wherein process activities are adapted during the course of a project based on measurements of process performance. Planning and costing such processes prior to the start of the project is impossible. (Controlling them will also be difficult.)

What Is an Estimate?

As a minimum, the estimator must compute the effort (cost) and duration (schedule) for the project's process activities, identify associated costs such as equipment, travel and staff training, and state the rationale behind the calculations (input values used, etc.). Estimation is closely tied to the details of the product's requirements and design (primarily the software architecture) and the activities of the chosen development process. These must be well understood to produce accurate estimates.

It also is highly desirable for the estimator to indicate the confidence in the reported values via ranges or standard deviations. The estimator also should try to state the assumptions and risks to highlight any areas of limited understanding related to the requirements, product design, or development process.

Estimation Methods

There are two basic classes of estimation methods: experience based estimation and parametric models. Each has weaknesses. Experience based estimation may be flawed because of obsolescence of the historical data used or because the estimators' memory of past projects is flawed. Parametric models typically have a particular "perspective." Some clearly fit a military standard development process (such as that defined by DODSTD2167A). Other models fit commercial development procedures. Estimators must choose models suited to their project environment and ensure that these models are correctly calibrated to the project environment. In spite of their intrinsic weaknesses, both classes of methods have their uses.

Survey of Past Work

The formal study of software estimating technology did not begin until the 1960s, although some earlier work was done on models of research and development by Peter Norden [6]. This section gives a chronological summary of the work in the field.

The 1960s

In the 1960s, while at RCA, Frank Freiman developed the concept of parametric estimating, and this led to the development of the PRICE model for hardware. This was the first generally available computerized estimating tool. It was extended to handle software in the 1970s.

The 1970s

The decade of the 1970s was a very active period. During this decade, the need to accurately predict the costs and schedules for software development became increasingly important and so began to receive more attention. Larger and larger systems were being built, and many past projects had been financial disasters. Frederick Brooks, while at IBM, described many of these problems in his book *The Mythical Manmonth* [7]. His book provides an entertaining but realistic account of the problems as perceived at that time.

During the 1970s, high-order languages such as FORTRAN, ALGOL, JOVIAL, and Pascal were coming into increasingly wider use but did not support reuse. Also, programming tools (other than compilers for the languages and simple text editors) were in very limited use. For these two reasons, systems were essentially built by hand from scratch. The cost models of this period thus emphasized new development.

Many authors during the 1970s analyzed project data using statistical techniques in an attempt to identify the major factors contributing to software development costs. Significant factors were identified using correlation techniques and were then incorporated in models using regression techniques. (Regression is a statistical method to predict values of one or more dependent variables from a collection of independent (predictor) variables. Basically, the model's coefficients are chosen to produce the "best possible" fit to actual, validated project data.) Such models are one form of cost estimating relation (CER). The prototypical model of this type is the constructive cost model (COCOMO) developed by Barry W. Boehm in the late 1970s and described in his classic book "Software Engineering Economics." [1] Various implementations of COCOMO continue to be widely used throughout the world. PRICE S, a software cost estimation model, was also developed in the late 1970s by Frank Freiman and Robert Park. The PRICE parametric models were the first generally available computerized cost estimation models. William Rapp programmed the models, among them PRICE S, to run on mainframe computers at RCA, making them available via a time-sharing (dialin) service.

A shortcoming of these 1970s models is that the independent variables were often "result measures" such as the size in lines of code. Such values are readily measured but only after the project has been completed. It is very difficult to predict the values of such variables before the start of the project.2 This means that many of the models, although based on statistical analyses of actual result data, were hard to use in practice since the values of the independent variables were hard to determine before the project team had analyzed the requirements and had prepared a fairly detailed design. Another shortcoming of such models is that they assume that software will be developed using the same process as was used previously. As we have seen, this assumption is becoming increasingly unrealistic.

At the end of the 1970s, Allan Albrecht and John Gaffney of IBM developed function point analysis (FPA) to estimate the size and development effort for management information systems [8,9]. Components of a system are classified into five types according to specific rules. Weights are computed for the components of each type based on characteristics of the component. These weights are proportional to the development effort needed to construct components of that type. The estimator counts the number of components of each type, multiplies these counts by the corresponding weight, sums these products and multiplies the sum by a factor to account for global system characteristics. The result is the "size" of the system measured in "function points." The estimator then uses the team's productivity (in function points per person month) to compute the development effort.

Later in the 1970's, two authors endeavored to define models based on theoretical grounds. Lawrence H. Putnam [10] based his Software Lifecycle Model (SLIM) on the NordenRayleigh curve plus empirical results from 50 U.S. Army projects. Although still in use, this model is not generally believed to be especially accurate by authors such as E. Conte [11]. One of the most criticized features is its prediction that development effort scales inversely as the fourth power of the development time, leading to severe cost increases for compressed schedules. SLIM also states that effort scales as the cube of the system size. (In contrast, COCOMO states that effort scales as size to the 1.2 power. COCOMO's relation between development time and effort is equivalent to Putnam's model, i.e., development time scales as the cube root of the effort. However, COCOMO models the increase of effort due to schedule compression (and schedule relaxation as well) in terms of the fractional offset from the unconstrained or "nominal" schedule. The amount of effort increase is 20 percent or less.)

Maurice H. Halstead [12] defined software size in terms of the number of operators and operands defined in the program and proposed relations to estimate the development time and effort. To obtain this size information before the start of a project was nearly impossible because a good understanding of the detailed design is not available until later. Subsequent work by E. Conte ([11], page 300) has shown that Halstead's relations are based on limited data, and Halstead's model is no longer used for estimation purposes. (Don Coleman, et al. [13] have recently reported some success in using it to predict the "maintainability" of software.)

The 1980s

During the 1980s work continued to improve and consolidate the best models. As personal computers (PCs) started to come into general use, many models were programmed. Several firms began to sell computerized estimating tools. Following the publication of the COCOMO equations in 1981, several tools that implemented COCOMO appeared during the latter half of the 1980s.

The DoD introduced the Ada programming language in 1983 [American National Standards Institute (ANSI) and DODSTD1815A1983] to reduce the costs of developing large systems. Certain features of Ada significantly impact development and maintenance costs, so Barry Boehm and Walker Royce defined a revised model called Ada COCOMO [14]. This model also addressed the fact that systems were being built incrementally in an effort to handle the inevitable changes in requirements.

Robert C. Tausworthe [15] extended the work of Boehm, Herd, Putnam, Walston and Felix, and Wolverton to develop a cost model for NASA's Jet Propulsion Laboratory. Tausworthe's model was further extended by Donald Reifer to produce the PCbased SOFTCOSTR model, which is now sold by Resource Calculations Inc. Randall W. Jensen [16] extended the work of Putnam by eliminating some of the undesirable behavior of Putnam's SLIM. Putnam's SLIM equation has development effort proportional to size (in source lines of code) cubed divided by development time to the fourth power. Jensen asserted that development effort is proportional to the square of the size divided by the square of the development time. Both Jenson and Putnam apply the constraint that effort divided by the cube of the development time is less than some constant (which is chosen based on product and project parameters). Jensen's equations reduce to equations that are close to those of COCOMO's "embedded mode," but the effect of various cost drivers is handled quite differently. The Jensen model is currently sold as the Software Estimation Model (SEM), part of the System Evaluation and Estimation of Resources (SEER) tool set. Daniel Galorath and co-workers continue to refine and market this model. (Version 4.0 was released in late 1994.)

FPA was extended by Capers Jones [17] to include the effect of computationally complex algorithms on development costs. Various PCbased tools implement these FPAbased methods, such as Function Point Workbench and Checkpoint from Software Productivity Research and FPXpert and Estimacs from Computer Associates International.

Charles Symons [18] proposed another revision of FPA to achieve the following major goals: reduce the subjectivity in dealing with files, make the size independent of whether the system was implemented as a single system or a set of related subsystems, and change the emphasis of function points away from gauging the value to users to predicting development effort. His method, called Mark II Function Points, bases the computation of size (functionality) on "logical transactions." Each processing activity done by the system is analyzed in terms of the number of data items input, referenced, and output. These are counted and weighted to compute the "size" in function points.

The 1990s

Because of the increasing diversity of software development processes, the 1990s are seeing renewed attention on developing improved cost models. The SEI started a small initiative on software estimation improvement in 1994. Of more interest is an effort lead by Barry Boehm, now at the University of Southern California, to revise and extend the COCOMO model, which has served the industry well for so long (over 15 years).

The new version of COCOMO, called COCOMO 2.0, is still being developed and will address the various types of development processes mentioned earlier. COCOMO 2.0 explicitly handles the availability of additional information in later stages of a project, the nonlinear costs of reusing software components, and the effects of several factors on the diseconomies of scale. (Some of these are the turnover rate of the staff, the geographic dispersion of the team, and the "maturity" of the development process as defined by the SEI.) The model also revises some coefficient values and eliminates discontinuities present in the old model (related to "development modes" and maintenance vs. adaptation). Barry Boehm is leading a group of industrial and academic affiliates to review the revised set of COCOMO equations and to collect data to evaluate hypotheses to select the significant parameters for COCOMO 2.0. Boehm and his co-workers expect to provide the refined equations, representing an improved and calibrated cost model, in late 1996.

An Approach to Improve Estimation

I recommend a four-pronged approach to improve software cost and schedule estimation within an organization. Since people are the key, three of these prongs relate to helping people apply the best available practices. These actions will enable the organization to use the best known software estimation practices to accurately cost and intelligently manage software projects during the remainder of the 1990s and into the next century.

Improving Estimating Technology

Since various efforts are underway to improve cost models, it is important to stay abreast of developments in the field via attendance at conferences and membership in professional societies. The primary professional societies for cost estimation include the International Society of Parametric Analysts (ISPA), the COCOMO User's Group, and the Society of Cost Estimating and Analysis. Meetings and conferences include ISPA conferences, the annual COCOMO Conference, and the annual European Software Cost Modeling Conference (ESCOM). These provide access to stateoftheart information on cost estimation and models in the United States and abroad. Lastly, books and journals provide another source of useful information. This information can be conveyed to the staff via the organization's Software Engineering Process Group (SEPG).

Defining the Process

No single estimating method is suited for every type of project, and moreover, each model has its own particular strengths and weaknesses. Each organization needs to define an estimating process suited to its needs using policies and assistance provided by the SEPG. The process defined by each particular organization is documented so it can be applied consistently, taught to others, reviewed for completeness and correctness, and improved. I advocate a documented estimating process that uses at least two methods with complementary strengths and weaknesses, coupled with independent reviews. For example, a parametric model and a manual method are typically used together.

Since there is much uncertainty about software size in the early stages of a project, I advocate regular review of progress, assumptions and product requirements during a project to detect changes and violations of the assumptions underlying the estimate. Proactive risk management is a part of this review process as well. Additional reviews may be triggered when unexpected events occur, such as new or changed requirements.

When conditions change, estimates are revised. Since more data becomes available as a project unfolds, different cost models and CERs are often used in the later stages of a project. These give more accurate predictions and provide managers with the best possible information on expected costs as the project unfolds.

Obtaining Management Sponsorship

Managers must lead the way to improve software cost estimation practices but are faced with many demands on their time and resources. Each organization, via its SEPG and its training program, can engender management support for improving software cost estimation in two ways. First, make managers aware of typical estimation problems and methods available to solve or, whenever possible, avoid these problems. This is best done with a short course that provides an overview of software estimation.

Second, the organization's SEPG should define standard measures for software projects and an integrated approach to collect these metrics. The information collected should be useful to both the manager and the estimator. Managers use the information for the daytoday management of their projects. Estimators use the data as inputs to CERs and to calibrate the estimation models. Another practical strategy is to have support organizations (configuration management, quality assurance, finance and accounting) collect and report many of the measures needed thereby reducing the burden on the software engineers and their managers.

Providing the Infrastructure

The organization should establish an infrastructure to advocate and sustain efforts to improve software cost estimation techniques. The primary means used is the organization's SEPG. The CMM developed by the SEI can be used as the guide for this program. The SEPG is assisted by the organization's training department, which conducts a comprehensive management training program, and organizational policies that relate to software development and project management.

The SEPG develops and maintains under configuration control the organization's defined process. This document contains policies that relate to software development and specifies requirements for processes acceptable to the organization and which comply with the criteria of the SEI's CMM. The defined processes are enacted via several sets of detailed procedures, each meeting the needs of particular business units within the organization. Of particular interest in the present context are procedures for metrics, cost estimation, project planning, and project tracking.

The SEPG (or related working groups) conducts activities that relate to improving software cost estimation practices. For example, a metrics working group would define metrics to track projects and gauge their progress and costs. As noted above, these standardized metrics are chosen to be useful to both cost estimators and project managers. Similarly, a tools working group would review available tools including estimation and planning tools to identify the "best of breed" for possible use by projects. The tools working group could also arrange licenses for selected tools, thereby making them available on a "try-before-buy" basis. A training working group would develop and deliver courses that teach managers and software engineers techniques to estimate and plan better.

Lastly, it may even be desirable to establish a software estimation assistance team (SEAT) to promote the adoption and effective use of improved cost estimation techniques within the organization. The SEAT would track the evolution of software cost estimation methods and models, identify those that are most useful for the organization's various business areas, and act as a technology transfer agent by revising training courses, updating portions of the organizational process definition, and mentoring projects that need help with software estimation. The SEAT would provide information and advice for proposals that have large software content. SEAT members also could review cost estimates for proposals and projects.

Some Predictions

All software estimation models continue to face the obstacle that no valid theoretical models of software development exist. Thus, there are no laws of "software physics" that can define logical constraints and relationships between various independent variables present in the project and product environment. Software development remains a highly intellectual activity and so is heavily dependent on human thought. Until psychology can build quantitative models of human problem solving, there is little hope of developing software physics. Consequently, cost estimation will remain an experimental science for the foreseeable future. Estimators must rely on judgment and intuition to define heuristic rules, then validate these via analysis of actual project data. Once the significant factors have been isolated, simplified models can be defined and calibrated for use by the estimators. The resulting models can then be used in industry. (This is what the COCOMO 2.0 project is doing.)

Practical Advice

As new software architectures, reusable components, development processes, and support tools are developed, new estimation methods will continue to appear. Each organization should endeavor to track the evolution and refinement of software estimation models in the literature and via conferences. It should also establish a means to identify and infuse proven techniques into active use throughout the organization. These actions will enable the organization to use the best known software estimation practices to accurately cost and intelligently manage software projects during the remainder of the 1990s and into the next century.

Richard D. Stutzke
Science Applications International Corp.
6725 Odyssey Drive
Huntsville, AL 358063301
Voice: 2059716624
Fax: 2059716550
e-mail: richard.d.stutzke@cpmx.saic.com

References

1. Boehm, Barry W., "Software Engineering Economics," PrenticeHall, 1981. Section 29.7 describes several models not discussed in this article as well as the models developed by Herd, Putnam, Walston, and Wolverton.

2. Selby, Richard, "Empirically Analyzing Reuse in a Production Environment" in "Software Reuse: Emerging Technology," W. Tracz, IEEE Computer Society Press, 1988, pp. 176189.

3. Gerlich, Rainer, and Ulrich Denskat, "A Cost Estimation Model for Maintenance and High Reuse," *Proceedings of the European Software Cost Modeling Conference*, 1994, Ivrea, Italy, May 1994.

4. Boehm, Barry W., Bradford Clark, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby, "Cost Models for Future Software Lifecycle Processes: COCOMO 2.0," *Annals of Software Engineering*, to be published. Also see the tutorial "COCOMO, Ada COCOMO, and COCOMO 2.0" by Barry Boehm in the *Proceedings of the Ninth International COCOMO Estimation Meeting*, *Los Angeles*, Calif. Oct. 67, 1994.

5. Selby, Richard, D. Schmidt, and J. Berney, "MetricDriven Analysis and Feedback Systems for Enabling Empirically-Guided Software Development," *Proceedings of the Thirteenth International Conference on Software Engineering*, Austin, Texas, May 1316, 1991, pp. 288298.

6. Norden, Peter V., "Curve Fitting for a Model of Applied Research and Development Scheduling," *IBM Journal of Research and Development*, Vol. 2, No. 3, July 1958.

7. Brooks, Frederick P., "The Mythical ManMonth," Addison Wesley, 1975. An updated and expanded edition was published in 1995.

8. Albrecht, Allan J., "Measuring Application Development Productivity," *Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium*, Oct. 1417, 1979.

9. Albrecht, Allan J., and John E. Gaftney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*, Vol. 9, No. 2, November 1983.

10. Putnam, Lawrence H., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering SE4*, July 1978, pp. 345361.

11. Conte, E., H. E. Dunsmore, and V. Y. Shen, "Software Engineering Metrics and Models," Benjamin Cummings, 1986.

12. Halstead, Maurice H., "Elements of Software Science," Elsevier, New York, 1977.

13. Coleman, Don, Dan Ash, Bruce Lowther, and Paul Oman, "Using Metrics to Evaluate Software System Maintainability," *IEEE Computer*, Vol. 27, No. 8, August 1994, pp. 4449.

14. Boehm, Barry W., and Walker Royce, "Ada COCOMO and the Ada Process Model," *Proceedings of the Third International COCOMO Users Meeting*, Software Engineering Institute, Pittsburgh, Pa., November 1987, plus refinements presented at the Fourth International COCOMO Users Group Meeting held in November 1988.

15. Tausworthe, Robert C., "Deep Space Network Estimation Model," Jet Propulsion Report 817, 1981.

16. Jensen, Randall W., "A Comparison of the Jensen and COCOMO Estimation Models," *Proceedings of the International Society of Parametric Analysts*, 1984, pp. 96106.

17. Jones, Capers, "The SPR Feature Point Method," Software Productivity Research, Inc., 1986.

18. Symons, Charles, "Software Sizing Estimating: Mark II FPA," Wiley, 1991.

Notes

1. There is an analogy here to the situation shortly after telephones were introduced in the United States. Someone predicted that soon the majority of the U.S. population would have to be employed as telephone operators to handle the workload of connecting calls. Instead, technology changed, and now everyone serves as their own operator, using direct dialing to connect nearly all of their calls.

2. Some methods have been developed to help estimators estimate size. These use analogies to similar completed projects whose costs are recorded in a database, or use various averaging techniques to elicit the consensus of experts (Delphi, PERT) or combinations thereof. One such combination, developed by George Bozoki of Lockheed, uses sophisticated statistical techniques in conjunction with historical data. Bozoki's model is sold as the Software Sizing Model by Galorath Associates.

3. Figure 1 is reproduced with permission from [1], page 311.

About the Author

Richard D. Stutzke is an employee owner of Science Applications International Corporation (SAIC) and has worked in the software field for over 30 years. He has developed a wide range of software including realtime satellite tracking systems and commercial accounting systems. He established and led SAIC's Corporate Software Process Group for its first two years. He currently supports software process improvement at the U.S. Army Missile Command's Software Engineering Directorate, defining processes for software transition, software sustainment, and technology transfer. He has written several papers on software cost estimation. He is the principal author of SAIC's software estimating course and has taught over 1,000 students since developing the course in 1990. He is presently writing a book on practical estimating techniques for software projects. From 1969 through 1972, he served in the U.S. Air Force. He was trained in physics and received a bachelor of science degree in physics from Purdue University and holds a master of science degree and a doctorate in nuclear physics from the University of Illinois.