# Software Estimation: Challenges and Research

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

## The Changing Nature of Software Development

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

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

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

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

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

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

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

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

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

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

## Promising Research

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

### *Size*

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

Table 1. *Ways to Acquire Software**

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

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

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

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

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

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

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

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

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

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

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

## Standardized Process Milestones

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

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

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

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

**Inception Readiness Review (IRR)**

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

**Life Cycle Objectives Review (LCO)**

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

**Life Cycle Architecture Review (LCA)**

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

**Unit Test Complete (UTC)**

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

**Initial Operational Capability (IOC)**

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

**Transition Readiness Review**

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

**Product Release Review (PRR)**

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

Table 2. *New Process Milestones*

## Theoretical Productivity Models

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

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

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

## Parametric Models

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

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

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

## Back to Basics

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

## References


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


## Notes

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

## About the Author


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

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