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:

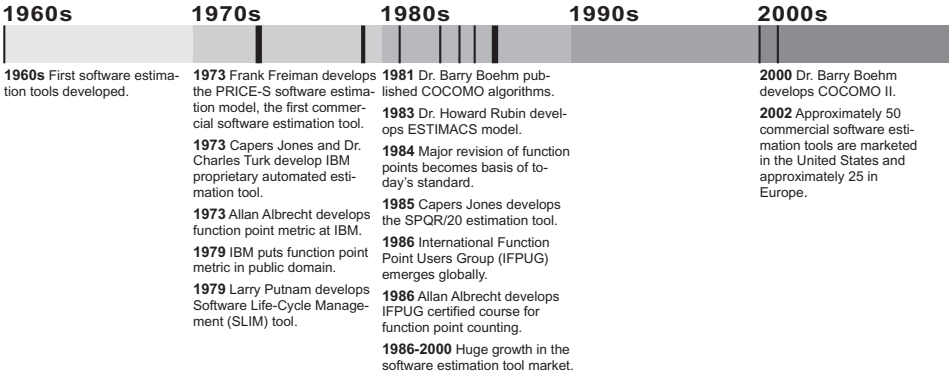
- Risk and value analysis.
- Estimation templates derived from historical data.
- Links to project management tools such as Artemis or Microsoft Project.
- Cost and time-to-complete estimates mixing historical data with projected data.
- Currency conversions for international projects.
- Inflation calculations 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