HOW SOFTWARE ESTIMATION TOOLS WORK

Version 5 – February 27, 2005

Abstract

Since the mid 1990's there have been about 50 commercial software cost estimation tools marketed in the United States and another 25 in Europe, although not all at the same time. Many of these tools are "black boxes" and their methods of operation are proprietary and regarded as trade secrets by their owners. However, there is a basic sequence of activities that must be carried out. This article discusses eight essential steps of accurate software estimation: 1) Sizing project deliverables; 2) Estimating quality and defect removal efficiency; 3) Selecting project activities; 4) Estimating staffing levels; 5) Estimating Effort; 6) Estimating costs; 7) Estimating schedules; 8) Estimating requirements growth during development.

The author is the designer of eight proprietary software estimating tools. His company has developed three commercial software estimating tools: SPQR/20™ in 1985, CHECKPOINT® (marketed under the name CHECKMARK® in the United Kingdom and Japan.) in 1989, and KnowledgePlan® in 1997. In addition, Software Productivity Research also performs assessments, baselines, and benchmarks on a global basis.

Capers Jones, Chief Scientist Emeritus
Software Productivity Research LLC.
Email CJones@spr.com
Web http://www.spr.com

INTRODUCTION

Software has achieved a bad reputation as a troubling technology. Large software projects have tended to have a very high frequency of schedule overruns, cost overruns, quality problems, and outright cancellations of large systems. While this bad reputation is often deserved, it is important to note that some large software projects are finished on time, stay within their budgets, and operate successfully when deployed.

The successful software projects differ in many respects from the failures and disasters. One important difference is how the successful projects arrived at their schedule, cost, resource, and quality estimates in the first place. It often happens that projects exceeding their planned schedules or cost estimates did not use state of the art methods for determining either their schedules or their costs. Although the project is cited for overruns, the root problem is inadequate planning and estimation.

From large-scale studies of the Patterns of Software Systems Failure and Success (Jones 95) usage of automated estimating tools, automated project scheduling tools, and automated defect estimating tools are strongly correlated with successful outcomes. Conversely, software failures tended to use casual and manual methods of arriving at initial estimates. Indeed, for many software failures there was no formal estimation at all

Because software estimation is a complex activity there is a growing commercial industry of companies that are marketing software estimation tools. Since the author and his company build commercial software estimation tools, we perform surveys of the estimating subindustry on an annual basis. As of calendar year 2000 there are approximately 50 commercial software estimating tools marketed in the United States, and another marketed 25 abroad. About one new software estimating tool has entered the commercial market almost every month since 1992.

History of the Commercial Software Estimation Industry

The software cost estimation market was created by researchers who were employed by large enterprises that built large and complex software systems: IBM, RCA, TRW, and the U.S. Air Force were the organizations whose research which led to the development of commercial cost estimating tools.

The software estimation business dates back to the early 1970's. In 1973 the PRICE-S software estimation model designed by Frank Freiman and his colleagues was put on the market as the first software commercial estimation tool in the United States, and perhaps the world. This product was originally marketed by the RCA corporation, but ownership has changed hands several times and it is now marketed by Lockheed Martin.

Also in 1973 the author and his colleague Dr. Charles Turk at IBM San Jose built IBM's first automated estimation tool for systems software, although this tool was proprietary and not put on the commercial market. In that same year, Allan Albrecht and his colleagues at IBM White Plains were developing the original version of the function point metric, which has become a major feature of modern software estimation.

In October of 1979 Allan Albrecht, then at IBM, gave a paper on function point metrics at a joint IBM/SHARE/GUIDE conference in Monterey, California. At the same time IBM put the structure of the function point metric into the public domain. Since function point metrics are now supported by more than 30 software estimation tools in the United States, this was a significant step.

In 1979, the second commercial estimation tool reached the U.S. market. This was the SLIM (software lifecycle management) tool developed by Larry Putnam of Quantitative Software Management. The original research for this tool was performed when Larry Putnam was a Colonel in the United States Air Force.

In 1981, Dr. Barry Boehm of the TRW corporation published his monumental book on Software Engineering Economics (Boehm, 81) which contained the essential algorithms of the "constructive cost model" or COCOMO. Since the COCOMO algorithms were described in print, many commercial estimating tools have been derived from the COCOMO estimation method. Still today in 2000 COCOMO remains the only software estimating model whose algorithms are openly published and not treated as trade secrets.

Because the original COCOMO algorithms were published and readily available, at least 25 COCOMO "clones" have been put on the market or made available. Some of these COCOMO derivatives include REVIC (revised COCOMO), SoftCost, SEER, BYL (Before You Leap), and many others.

Dr. Boehm and his colleagues have also developed a new version of COCOMO called COCOMO II, which is to be supported by a new book in the summer of 2000. COCOMO II expands the capabilities of the original model and can estimate applications using modern development methods. (Boehm et al 2000).

Continuing with the history of estimation, in 1983, Dr. Howard Rubin's ESTIMACS model reached the commercial market. ESTIMACS was a derivative of some of IBM's internal estimating methodology for information systems projects and supported an early form of function point metric prior to IBM's major revision of function points in 1984, which is the basis of today's standard function point.

In 1985 the author's SPQR/20 (Software productivity, quality, and reliability) estimation tool reached the commercial market. The SPQR/20 estimation tool was the first commercial software estimator built explicitly to support function point metrics. It was also the first software estimation tool to include full sizing logic, and the first to include "backfiring" or direct conversion from lines of code (LOC) metrics to function point metrics and vice versa. This tool was also the first to integrate software cost estimation, software risk analysis, and software quality estimation in a single tool.

In 1986 usage of function point metrics had grown so rapidly that the non-profit International Function Point Users Group (IFPUG) started to become a global organization. Also in 1986, Allan Albrecht retired from IBM and joined Software Productivity Research where he developed the first IFPUG-certified course for function point counting.

The author's SPQR/20 tool was followed in 1989 by the more powerful CHECKPOINT tool which integrated software process assessments, software measurements, IFPUG function point support, and software task-level estimation into a single tool. The CHECKPOINT estimation tool was the first to move beyond phase-level estimation and achieve activity-level and task-level cost estimation.

In 1997, the author's company released the KnowledgePlan software cost estimating tool. This tool was a native Windows application, and included direct bi-directional interfaces with Microsoft Project and more generic interfaces to other project management tools. This tool also supports varying levels of granularity from project-level macro estimation down to task-level micro estimation.

These pioneering commercial software estimating tools plus the invention and publication of function point metrics mark the emergence of the software cost estimation industry. From 1986 to 2000, new commercial software estimation tools have occurred so rapidly that it is now difficult to keep track of the growth of the estimating industry, and at least 50 software cost estimation tools are marketed in the United States alone. The major features of commercial software estimation tools include these attributes:

- 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 modern languages such as Java and Visual Basic
- Support for web-based applications
- Support for including commercial off the shelf software (COTS)

Most of the commercial estimating tools include these basic functions. In addition, some estimating tools also include more advanced functions such as:

- Quality and reliability estimation
- Risk and value analysis
- Measurement modes for collecting historical data
- Cost and time to complete estimates mixing historical data with projected data
- Support for software process assessments
- Statistical analysis of multiple projects and portfolio analysis

Modern software cost estimating tools are now capable of serving a variety of important project management functions. In addition, the accuracy and precision of such tools is now high enough to merit their use for business agreements such as contracts and outsource agreements.

The software estimation tool market has proven to be somewhat volatile. Some of the earlier estimation tools such as Estimacs, REVIC, SPQR/20 and the original COCOMO are no longer used or marketed, while newer tools such as KnowledgePlan and COCOMO II are entering the market.

One of the challenges of the estimating tool market is to stay current with software development and maintenance technologies. To name but a few, software estimation tools must be able to support new programming languages such as Javascript and HTML, and new methods such as the Agile methods, the Six-Sigma approach, and all levels of the Software Engineering Institute's capability maturity model (CMM).

Software Cost Estimation Tools and Project Management Tools

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.

However, the family of project management tools are general purpose in nature, and do not include specialized software sizing and estimating capabilities as do the software cost estimating tools. Neither do these general project management tools deal with quality issues such as defect removal efficiency. Project management tools are useful, but software requires additional capabilities to be under full management control.

There are a host of managerial functions that standard project management tools don't deal with in depth, such as contract management, personnel management, assessments, size estimating, quality estimating, defect tracking, and the like.

The software cost estimation industry and the project management tool industry originated as separate businesses, with project management tools appearing about in the 1960's, about 10 years before software cost estimating tools. Although the two were originally separate businesses they are now starting to merge together technically, and also via mergers and acquisitions.

Project management tools are an automated form of several management aids developed by the Navy for controlling large and complex weapons systems: the "program evaluation and review technique" (PERT), critical path analysis, resource leveling, and the classic Gantt charts.

Project management tools did not originate for software, but rather for handling very complex scheduling situations where hundreds or even thousands of tasks need to be determined and sequenced, and where dependencies such as the completion of a task might affect the start of subsequent tasks. Project management tools are general-purpose in nature, and can be applied to any kind of project: building an aircraft, constructing an office building, or a software project.

Project management tools have no built-in expertise regarding software, as do software cost estimating tools. For example, if you wish to explore the quality and cost impact of

an object-oriented programming language such as Objective C a standard project management tool is not the right choice.

By contrast, many software cost estimating tools have built-in tables of programming languages and will automatically adjust the estimate based on which language is selected for the application.

Since software cost estimating tools originated about 10 years after commercial project management tools, the developers of software cost estimating tools seldom tried to replicate project management functions such as construction of detailed PERT diagrams or critical path analysis. Instead, the cost estimation tools would export data so that the final schedule could be adjusted via the project management tool. Thus bi-directional interfaces between software cost estimating tools and project management tools are now standard features in the commercial estimation market.

Macro Estimation and Micro Estimation

There are two major varieties of software estimation logic in the commercial software estimation tools. One form is called "macro estimation" or sometimes "top down estimation." The other form is called "micro estimation" or sometimes "bottom up" estimation.

In the form of estimation termed macro estimation the estimating equations are aimed at complete software projects. Once the effort and schedule for the overall project is predicted, then the overall estimate is divided into relative amounts for each phase such as requirements, design, coding, etc.

A simple example of a macro estimation equation would be one to determine the overall schedule of a software project from requirements to deployment. Raising the function point total of a software application to the 0.4 power will give a rough approximation of the number of calendar months from requirements to delivery.

In the form of estimation termed "micro estimation" the estimating equations deal with specific activities such as requirements, design, coding, unit test, integration, user documentation, and so on.

Rather than a single equation for the entire project, the estimate would be built up from the overall aggregation of partial estimates for each activity or each deliverable.

Of the two forms, macro estimation is easier to perform but has the problem that any errors in the estimate will propagate throughout every phase or activity. Micro estimation is more complex since each activity is estimated independently.

The main advantage of micro estimation is that errors tend to be localized to specific deliverables. With macro estimation, an error of say 15% will be present in every phase since the entire project is the primary unit of estimation. With micro estimation, there may be an error of 15% in a specific activity such as requirements, but the errors do not flow from activity to activity since each activity is estimated separately. Also, micro-estimation

can drop below phase-level estimation, the normal limit for macro estimates, and achieve activity-level and task-level estimation capability.

Macro estimation is useful for quick preliminary estimates, but for estimates with serious business purposes such as software development contracts, micro estimation is the method with the highest precision.

Three Fundamental Equations of Software Estimating

Although software cost estimating is a very difficult intellectual problem, there are three fundamental equations that are rather straightforward. These equations make use of important estimating concepts that are linked together for estimation: 1) *assignment scopes*, 2) *production rates*, 3) *schedules*.

The "assignment scope" is the amount of some kind of work for which one person will normally be responsible. The "production rate" is the amount of work which one person can complete in a given time period such as an hour, a week, a month, or a year. The schedule is the effort required for a piece of work divided by the staff available to perform it. An example can clarify the three concepts.

Suppose you are managing an application of 100,000 lines of source code using the C programming language. A typical assignment scope for an average C programmer is roughly 10,000 lines of source code, so this application might require 10 programmers.

Journeyman C programmers have average coding production rates of about 2,000 lines of source code per month, so the coding of the application will take about 50 person-months of effort.

If the effort for coding the project is 50 person-months and there are 10 programmers assigned, then the schedule for coding will be 5 calendar months.

Note that the logic of assignment scopes and production rates can be expressed in almost any metric: lines of code, function points, pages of text, test cases, or whatever. The normal sequence of applying these estimating equations is:

1. Size of deliverable / assignment scope = staff
2. Size of deliverable / production rate = effort
3. Effort / staff = schedule

The equations are simple in principle, but very tricky in real life. The difficulty is not in the equations themselves, but in knowing the exact assignment scopes and production rates with enough precision for the equations to yield useful results. It is the ability to adjust the fundamental equations in response to varying degrees of staff skill, to different tools, to different methodologies, and to different programming languages that make commercial software estimating tools valuable. This same kind of information is so difficult to acquire and so valuable that the estimating tool vendors almost universally treat adjustment data as proprietary information or as trade secrets.

For example, suppose your programmers are "top guns" and among the best in the industry. In this situation, the assignment scope might be 20,000 lines of source code and the production rate might be 4,000 lines of code per month. Under these modified assumptions the 100,000 LOC example project might take only five programmers and 25 months of effort although the schedule would still be five calendar months.

Conversely, suppose you are building the application with a staff of novices who have no experience in the application and are just learning the language too. In this case, the same project might have an assignment scope of only 7,500 lines of source code and the production rate might only be 1,500 lines of code per month. Under these assumptions a staff of 13 would be needed, the project would take 67 months of effort, and the schedule would be roughly five months.

These examples are hypothetical and illustrate some of the many variations which can occur in real life. Performance ranges of software engineers are so broad that it is necessary to know the capabilities of the actual team in order to match the estimate with the reality of the situation. However the fundamental equations can be used across all variations, so long as the estimating tool can be adjusted to match the variations which occur.

As can be seen there are strong linkages between measurement and estimation. Companies that measure software projects carefully have solid empirical data available on the specific assignment scopes, production rates, and schedules of the projects that they have developed. Companies that do not measure, on the other hand, must depend upon subjecting hunches unless they utilize one or more of the commercial software cost estimating tools.

Even if such tools are used, it is still useful to calibrate them by comparing their results against measured data. Thus measurement of software projects is quite valuable for minimizing future problems with estimation.

EIGHT STEPS OF SOFTWARE COST ESTIMATION

The algorithms of the more powerful commercial software cost estimating tools are usually regarded as trade secrets. Only the algorithms for the COCOMO model have been fully published and put into the public domain. However, all software estimating tools typically perform the six generic functions discussed here: 1) Sizing project deliverables, 2) Estimating defect levels and removal efficiency; 3) Selecting project activities; 4) Estimating staffing levels; 5) Estimating Effort; 6) Estimating costs; 7) Estimating schedules; 8) Estimating requirements changes during development.

This is the sequence of activities used by all of the software estimating tools developed by Software Productivity Research: SPQR/20, CHECKPOINT, and KnowledgePlan.

Step 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. The older software cost estimating tools such as COCOMO did not include sizing logic, and size information had to be provided by the user.

However, the invention of function point metrics has made full sizing logic for all deliverables a standard feature of estimating tools starting with SPQR/20 in 1985. As of 1998 sizing is a standard feature of commercial software cost estimating tools, and a variety of sizing methods are now included, such as:

1. Sizing based on function point metrics
2. Sizing based on lines of code (LOC) metrics
3. Sizing based on object-oriented metrics
4. Sizing based on analogy with past projects
5. Sizing based on attributes such as screens and reports created

Here are selected size examples, drawn from systems, MIS, military, and commercial software domains. In this context, "systems" software is that which controls physical devices such as computers or telecommunication systems. MIS software stands for "management information systems" and refers to the normal business software used by companies for internal operations. Military software constitutes all projects which are constrained to follow various military standards. Commercial software refers to ordinary packaged software such as word processors, spreadsheets, and the like.

Table 1: Number of Pages Created Per Function Point for Software Projects

	Systems Software	MIS Software	Military Software	Commercial Software
User Requirements	0.45	0.50	0.85	0.30
Functional specifications	0.80	0.55	1.75	0.60
Logic specifications	0.85	0.50	1.65	0.55
Test plans	0.25	0.10	0.55	0.25
User tutorial documents	0.30	0.15	0.50	0.85
User reference documents	0.45	0.20	0.85	0.90
Total document set	3.10	2.00	6.15	3.45

This kind of sizing for software documentation is now a standard feature of several commercial software cost estimating tools. At least one commercial software estimating tool can even predict the number of English words in the document set, and also the numbers of diagrams that are likely to be present, and can change the page count estimates based on type size. Indeed, it is now possible to estimate the sizes of text-based documents in several national languages (i.e. English, French, German, Japanese, etc.) and even to estimate translation costs from one language to another.

The second major sizing capability associated with function point metrics is the ability to predict source code size for any programming language, or even for applications that use two or more languages at the same time such as COBOL and SQL or Ada and Jovial.

There are far too many languages to do more than illustrate the concept, but source code sizing consists of predicting the number of logical statements that will be needed to encode one function point.

Table 2: Ratios of Logical Source Code Statements to Function Points

Language	**Nominal Level**	**Source Statements Per Function Point**		
		Low	**Mean**	**High**
Basic assembly	1.00	200	320	450
Macro assembly	1.50	130	213	300
C	2.50	60	128	170
FORTRAN	3.00	75	107	160
COBOL	3.00	65	107	150
PASCAL	3.50	50	91	125
ADA 83	4.50	60	71	80
C++	6.00	30	53	125
Ada 9X	6.50	28	49	110
SMALLTALK	15.00	15	21	40

SQL	27.00	7	12	15

Most commercial software estimating tools now include code sizing derived from function points. For greater accuracy, it is necessary to include adjustments for both complexity and for reusable code. It is also necessary to deal with applications that include multiple programming languages. However, all of these capabilities are now available in commercial software estimating tools.

Another useful sizing capability associated with function point metrics is the ability to predict the number of test cases that are likely to be created for the application. Here too, there are major differences in test case numbers by industry, with military software and systems software producing much larger totals of test cases than information systems. There is now at least preliminary data available for more than 20 kinds of standard kinds of testing. (In addition, the effectiveness of formal design and code inspections can also be predicted, although these methods do not use formal test cases.) Following are some representative examples:

Table 3: Number of Test Cases Created per Function Point

	Systems Software	MIS Software	Military Software	Commercial Software
Unit test	0.30	0.20	0.50	0.30
New function test	0.35	0.25	0.35	0.25
Regression test	0.30	0.10	0.30	0.35
Integration test	0.45	0.25	0.75	0.35
System test	0.40	0.20	0.55	0.40
Total test cases	1.80	1.00	2.45	1.65

Test case sizing logic is now available in commercial software estimating tools. The total number of kinds of testing covered is now up to about a dozen discrete forms. The sizing logic for test cases in the best commercial software estimating tools also includes adjustments for enhancements rather than for new projects, for complexity factors, and for the impact of ancillary tools such as test case generators and test coverage analysis tools.

Step 2: Estimating Defects and Defect Removal Efficiency Levels

A key aspect of software cost estimating is predicting the time and effort that will be needed for design reviews, code inspections, and all forms of testing. In order to estimate defect removal costs and schedules, it is necessary to know about how many defects are likely to be encountered.

The typical sequence is to estimate defect volumes for a project, and then to estimate the series of reviews, inspections, and tests that the project utilize. The defect removal efficiency of each step will also be estimated. The effort and costs for preparation, execution, and defect repairs associated with each removal activity will also be estimated.

Table 4 illustrates the overall distribution of software errors among the same six project types shown in table 1. In table 4, bugs or defects are shown from five sources: requirements errors, design errors, coding errors, user documentation errors, and "bad fixes." A "bad fix" is a secondary defect accidentally injected in a bug repair. In other words, a bad fix is a failed attempt to repair a prior bug that accidentally contains a new bug. On average about 7% of defect repairs will themselves accidentally inject a new defect, although the range is from less than 1% to more than 20% bad fix injections.

(The data in table 4, and in the other tables in this report, are based on a total of about 12,000 software projects examined by the author and his colleagues circa 1984-2004. Additional information on the sources of data can be found in (Jones 1996), (Jones 1998), and (Jones 2000). See also (Kan 2003)).

Table 4: Average Defect Potentials for Six Application Types (Data expressed in terms of "defects per function point"

	Web	MIS	Outsource	Commercial	System	Military	Average
Requirements	1.00	1.00	1.10	1.25	1.30	1.70	1.23
Design	1.00	1.25	1.20	1.30	1.50	1.75	1.33
Code	1.25	1.75	1.70	1.75	1.80	1.75	1.67
Documents	0.30	0.60	0.50	0.70	0.70	1.20	0.67
Bad Fix	0.45	0.40	0.30	0.50	0.70	0.60	0.49
TOTAL	4.00	5.00	4.80	5.50	6.00	7.00	5.38

Table 4 presents approximate average values, but the range for each defect category is more than 2 to 1. For example, software projects developed by companies who are at level 5 on the capability maturity model (CMM) might have less than half of the potential defects shown in table 3. Similarly, companies with several years of experience with the "Six Sigma" quality approach will also have lower defect potentials than those shown in table 3. Several commercial estimating tools make adjustments for such factors.

A key factor for accurate estimation involves the removal of defects via reviews, inspections, and testing. The measurement of defect removal is actually fairly straight forward, and many companies now do this. The U.S. average is about 85%, but leading companies can average more than 95% removal efficiency levels (Jones 1997).

It is much easier to estimate software projects that use sophisticated quality control and have high levels of defect removal in the 95% range. This is because there usually are no disasters occurring late in development when unexpected defects are discovered. Thus projects performed by companies at the higher CMM levels or by companies with extensive six-sigma experience for software often have much greater precision than average.

Table 5 illustrates the variations in typical defect prevention and defect removal methods among the six domains already discussed. Of course many variations in these patterns can occur. Therefore it is important to adjust the set of activities, and their efficiency levels, to match the realities of the projects being estimated. However, since defect removal in total has been the most expensive cost element of large software applications for more than 50 years, it is not possible to achieve accurate estimates without being very thorough in estimating defect removal patterns.

(The cumulative efficiency values in table 5 are calculated as follows. If the starting number of defects is 100, and there are two consecutive test stages which each remove 50% of the defects present, then the first test will remove 50 defects and the second test will remove 25 defects. The cumulative efficiency of both tests is 75%, because 75 out of a possible 100 defects were eliminated.)

Table 5: Patterns of Defect Prevention and Removal Activities

	Web	MIS	Outsource	Commer.	System	Military
Prevention Activities						
Prototypes	20.00%	20.00%	20.00%	20.00%	20.00%	20.00%
Clean Rooms					20.00%	20.00%
JAD sessions		30.00%	30.00%			
QFD sessions					25.00%	
Subtotal	*20.00%*	*44.00%*	*44.00%*	*20.00%*	*52.00%*	*36.00%*
Pretest Removal						
Desk checking	15.00%	15.00%	15.00%	15.00%	15.00%	15.00%
Requirements review			30.00%	25.00%	20.00%	20.00%
Design review			40.00%	45.00%	45.00%	30.00%
Document review				20.00%	20.00%	20.00%
Code inspections				50.00%	60.00%	40.00%
Ind. Verif. and Valid.						20.00%
Correctness proofs						10.00%
Usability labs				25.00%		
Subtotal	*15.00%*	*15.00%*	*64.30%*	*89.48%*	*88.03%*	*83.55%*
Testing Activities						
Unit test	30.00%	25.00%	25.00%	25.00%	25.00%	25.00%
New function test		30.00%	30.00%	30.00%	30.00%	30.00%
Regression test			20.00%	20.00%	20.00%	20.00%
Integration test		30.00%	30.00%	30.00%	30.00%	30.00%
Performance test				15.00%	15.00%	20.00%
System test		35.00%	35.00%	35.00%	40.00%	35.00%
Independent test						15.00%
Field test				50.00%	35.00%	30.00%
Acceptance test			25.00%		25.00%	30.00%
Subtotal	*30.00%*	*76.11%*	*80.89%*	*91.88%*	*92.69%*	*93.63%*
Overall Efficiency	52.40%	88.63%	96.18%	99.32%	99.58%	99.33%

Number of Activities	3	7	11	14	16	18

Table 5 oversimplifies the situation, since defect removal activities have varying efficiencies for requirements, design, code, documentation, and bad fix defect categories. Also, bad fixes during testing will be injected back into the set of undetected defects.

The low efficiency of most forms of defect removal explains why a lengthy series of defect removal activities is needed. This in turn explains why estimating defect removal is critical for overall accuracy of software cost estimation for large systems. Below 1000 function points, the series of defect removal operations may be as few as three. Above 10,000 function points the series may include more than a dozen kinds of review, inspection, and test activity.

Step 2: Selecting Project Activities

Once the size of various deliverables has been approximated the next step is to determine which specific activities will be carried out for the project being estimated. This is one of the major areas where software cost estimating tools excel. Modern cost estimating tools can analyze the nature, size, and class of the application being estimated and automatically select the most likely set of activities.

There are some 25 common activities that might be performed for a software project, but only large military applications will normally perform all 25. Table 6 illustrates some of the variances in activity patterns based on the class of software project for six different subindustries. (Table 5 assumes applications larger than 1000 function points in size).

Table 6: Software Development Activities Associated with Six Subindustries (Data indicates the percentage of staff-months by activity)

Activities Performed	End-User	MIS	Outsource	Commer.	System	Military
01 Requirements		7.50%	9.00%	4.00%	4.00%	7.00%
02 Prototyping	10.00%	2.00%	2.50%	1.00%	2.00%	2.00%
03 Architecture		0.50%	1.00%	2.00%	1.50%	1.00%
04 Project plans		1.00%	1.50%	1.00%	2.00%	1.00%
05 Initial design		8.00%	7.00%	6.00%	7.00%	6.00%
06 Detail design		7.00%	8.00%	5.00%	6.00%	7.00%
07 Design reviews			0.50%	1.50%	2.50%	1.00%
08 Coding	35.00%	20.00%	16.00%	23.00%	20.00%	16.00%
09 Reuse acquisition	5.00%		2.00%	2.00%	2.00%	2.00%
10 Package purchase		1.00%	1.00%		1.00%	1.00%
11 Code inspections				1.50%	1.50%	1.00%
12 Ind. Verif. & Valid.						1.00%
13 Configuration mgt.		3.00%	3.00%	1.00%	1.00%	1.50%
14 Formal integration		2.00%	2.00%	1.50%	2.00%	1.50%
15 User documentation	10.00%	7.00%	9.00%	12.00%	10.00%	10.00%
16 Unit testing	40.00%	4.00%	3.50%	2.50%	5.00%	3.00%
17 Function testing		6.00%	5.00%	6.00%	5.00%	5.00%
18 Integration testing		5.00%	5.00%	4.00%	5.00%	5.00%

19 System testing		7.00%	5.00%	7.00%	5.00%	6.00%
20 Field testing				6.00%	1.50%	3.00%
21 Acceptance testing		5.00%	3.00%		1.00%	3.00%
22 Independent testing						1.00%
23 Quality assurance			1.00%	2.00%	2.00%	1.00%
24 Installation/training		2.00%	3.00%		1.00%	1.00%
25 Project management		12.00%	12.00%	11.00%	12.00%	13.00%
Total	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Activities	5	16	20	21	22	25

As can easily be seen, the variations in activities typically performed can affect overall costs, schedules, and productivity rates by significant amounts: more than 200% differences in work effort has been observed for projects of exactly the same size, due to variations in the work performed.

Step 4: Estimating Staffing Levels

Although staffing, effort, costs, and schedules are all important for the final estimate, the normal place to start estimating is with staffing levels. The fundamental equation for determining staff is:

Size of deliverable / assignment scope = staff.

Commercial software cost estimating tools apply this fundamental staffing equation in a wide variety of forms, including but not limited to:

Pages of specifications	/	assignment scope	=	analysts
Lines of source code	/	assignment scope	=	programmers
Test cases	/	assignment scope	=	testers
Pages of user manuals	/	assignment scope	=	technical writers
Number of employees	/	assignment scope	=	managers

The examples just shown make use of natural metrics that define the actual material to be created such as pages or code. The staffing equation can also be used with the synthetic function point metric. In fact the equations can be utilized with any metric if it can express the amount of work normally assigned to individual technical workers.

To illustrate these rules, typical assignment scopes for technical writers are about 150 pages. Assignment scopes for programmers are about 5000 source code statements. Typical assignments for test personnel are about 120 test cases. The typical number of employees reporting to a manager averages about 8. Of course, there are broad ranges with all such values. For example the author has noted departments where only three employees reported to a manager, while the upper value was about 25 employees reporting to a manager.

Adding to the difficulty of handling staffing, no fewer than 35 kinds of occupation groups may be found on large software projects. Thus it is necessary to determine staffing levels not only for software engineers, but for a host of other occupations such as quality assurance, technical writing, data base administration, measurement and metrics, testing, configuration control, and many others.

Step 5: Estimating Software Effort

The term "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 1000 function points utilize "hours" for expressing effort, but the larger projects in excess of 10,000 function points normally utilize "months" as the unit of measure.

Note that every company, and every country too, will have a characteristic profile of the number of effective work hours in a normal business day and the number of work days in a business year. It is very important to use local values for these parameters, since the variations in these topics are extreme. For example, in the United States the nominal work

week is 5 days of 8 hours each, or 40 hours in all. Yet the number of effective work hours per day is usually only about 6 due to coffee breaks, staff meetings, etc. The number of work days per year will vary with vacations and sick leave, but averages roughly 220 days per year in the United States. However, for Europe vacation periods are longer, while for other countries such as Mexico and Japan, vacation periods are shorter than in the United States. The fundamental equation for estimating effort is:

Size of deliverable / production rate = staff effort

Here too this basic equation is used in a variety of forms including but not limited to:

Pages of specifications	/	production rate	=	analyst months
Lines of source code	/	production rate	=	programmer months
Test cases	/	production rate	=	testing months
Defects found	/	production rate	=	rework months
Pages of user manuals	/	production rate	=	writing months

Note also that the use of "months" in this example is simply to indicate a work period. Estimates can be expressed in terms of work hours, work weeks, work months, or even work years.

Some typical examples of these rule are as follows: Specifications are normally prepared at a rate of about 20 pages per analyst month while technical writing proceeds at a rate of about 50 pages per month. Coding takes place at a rate of about 1000 source code statements per month. Here too there are very broad ranges and so average values need to be adjusted to match the actual project and team characteristics.

These adjustment factors for production rates are a critical portion of software estimating tools. For example, the author's CHECKPOINT and KnowledgePlan estimating tools contain several hundred rules just for adjusting production rates for various activities. Our rules are proprietary, as they are with other estimating tool vendors, but they deal with how production rates vary in response to these factors:

- Staff experience
- Type of software
- Size of software project
- Creeping requirements
- Methodologies used
- Design approach used
- Programming languages used
- Reusable materials available
- Unpaid overtime

For any given software activity, the measured range between the best performance and the worst performance is about 1000%. This is far too broad a range for "average" values to

be useful for serious estimation. Therefore knowledge of how to adjust production rates in response to various factors is the true heart of software estimation.

This kind of knowledge can only be determined by accurate measurements and multiple regression of analysis of real software projects. This explains why software estimating 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.

Step 6: 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 * (salary + burden) = cost

One 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.

The growing shortage of software personnel in the United States and elsewhere is driving salaries to unimaginable levels. Indeed, software engineers with skills in critical topics can command salaries of more than $100,000 in tight market areas such as the "Silicon Valley" region of California or Cambridge, Massachusetts.

For cost estimating purposes, "industry average" compensation levels are unsafe and the estimate should use the local compensation rates within the enterprise doing the work.

There are even greater variations in the burden rates or overhead structures which companies apply in order to recover expenses such as rent, mortgages, taxes, benefits, indirect personnel, and the like.

The burden rates in the United States can vary from less than 15% for small home-based enterprises to more than 300% for major corporations. Unlike basic compensation rates which are widely published, burden and overhead rates are not readily available. Indeed, some companies regard their burden rate structures as proprietary information which should not be released at all.

When the variance in basic staff compensation is compounded with the variance in burden rates, the overall cost differences are notable indeed. For example, assume that a small software company in a rural area is building an application of 100 function points in size for their own use. Assume that the average compensation of the programming staff is $36,000 per year and the burden rate is 50% which would bring the total cost structure up to $54,000 per year. If the project in question required 5 person months of effort, then

project would cost $22,500 or $225 per function point using the fully burdened cost structure.

Now assume that the same project was developed in a major urban area such as New York or San Francisco. Here the average compensation for the programming staff might be $80,000 per year, and the burden rate might be 100%. Under this cost structure the annual burdened personnel costs would be $160,000 per year. Thus assuming the same 5 person months of effort, the same 100 function point project would cost $66,666 or $666 per function point due to variations in compensation and burden rates.

With cost ranges such as those for projects that took exactly the same amount of programming staff effort, it is easy to see why figures such as "average cost per function point" should not be used without substantial adjustments for local conditions.

The cost situation is made more difficult by several other factors that need to be considered:

- Paid overtime for critical portions of rush projects
- Inflation rates for long-range projects spanning several years
- Currency exchange rates for international projects
- Special fees or costs outside the burden rate

Determining the true cost structure of a software project is a very complicated piece of work. The ranges of software costs are so broad that it is very hazardous to compare costs such as "cost per function point" across industries or geographic areas.

Step 7: Estimating Software Schedules

The fundamental equation for estimating the schedule of any given software development activity is:

Effort / staff = time period

Using this 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.

However in real life schedule estimating is one of the most difficult parts of the software estimation process and many highly complex topics must be dealt with, such as:

- Dependencies of one activity upon previous activities
- Overlap or concurrency of activities
- The critical path through the sequence of activities
- Less than full-time availability of staff
- 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

It is at this final point of determining software schedules that software cost estimating tools and project management tools come together. The normal mode of operation is that the software cost estimating 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.

As mentioned earlier, this synergistic relationship between software cost estimating tools and project management tools has become the norm and most commercial software cost estimating tools can export data to project management tools such as Artemis, Microsoft Project, Primavera, Time Line, or Project Managers Workbench.

Step 8: Estimating Requirements Changes During Development

One important aspect of estimating is dealing with the rate at which requirements "creep" and hence make projects grow larger during development. Fortunately, function point metrics allow direct measurement of the rate at which this phenomenon occurs, since both the original requirements and changed requirements will have function point counts.

Changing requirements can occur at any time, but the data in table 7 runs from the end of the requirements phase to the beginning of the coding phase. This time period usually reflects about half of the total development schedule. Table 7 shows the approximate monthly rate of creeping requirements for six kinds of software, and the total volume of change that might be anticipated:

Table 7: Monthly Rate of Changing Requirements for Six Application Types (From end of requirements to start of coding phases)

	Web	MIS	Outsource	Commercial	System	Military	Average
Monthly Rate	4.00%	2.50%	1.50%	3.50%	2.00%	2.00%	2.58%
Months	6.00	12.00	14.00	10.00	18.00	24.00	14.00
TOTAL	24.00%	30.00%	21.00%	35.00%	36.00%	48.00%	32.33%

For estimates made early in the life cycle, several estimating tools can predict the probable growth in unplanned functions over the remainder of the development cycle. This knowledge can then be used to refine the estimate, and to adjust the final costs in response.

Of course, the best response to an estimate with a significant volume of projected requirements change is to improve the requirements gathering and analysis methods. Thus

projects that use prototypes, joint application design (JAD), requirements inspections, and other sophisticated requirements methods can reduce later changes down to a small fraction of the values shown in table 7. Indeed, the initial estimates made for projects using JAD will predict reduced volumes of changing requirements.

Summary and Conclusions

Software estimating is simple in concept, but difficult and complex in reality. The difficulty and complexity required for successful estimates exceeds the capabilities of most software project managers to produce effective manual estimates.

The commercial software estimating tools can often outperform 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 estimating tools coupled with software project management tools, under the careful guidance of experienced software project managers and estimating specialists.

References and Readings on Software Cost Estimation

Albrecht, Allan; AD/M Productivity Measurement and Estimate Validation; IBM Corporation, Purchase, NY; May 1984.

Barrow, Dean, Nilson, Susan, and Timberlake, Dawn; Software Estimation Technology Report; Air Force Software Technology Support Center; Hill Air Force Base, Utah; 1993.

Boehm, Barry Dr.; Software Engineering Economics; Prentice Hall, Englewood Cliffs, NJ; 1981; 900 pages.

Brooks, Fred; The Mythical Man Month; Addison-Wesley, Reading, MA; 1995; 295 pages.

Brown, Norm (Editor); The Program Manager's Guide to Software Acquisition Best Practices; Version 1.0; July 1995; U.S. Department of Defense, Washington, DC; 142 pages.

Chidamber, S.R. and Kemerer, C.F.: "A Metrics Suite for Object Oriented Design"; IEEE Transactions on Software Engineering; Vol. 20, 1994; pp. 476-493.

Chidamber, S.R., Darcy, D.P., and Kemerer, C.F.: "Managerial Use of Object Oriented Software Metrics"; Joseph M. Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA; Working Paper # 750; November 1996; 26 pages.

Conte, S.D., Dunsmore, H.E., and Shen, V.Y.; Software Engineering Models and Metrics; The Benjamin Cummings Publishing Company, Menlo Park, CA; ISBN 0-8053-2162-4; 1986; 396 pages.

DeMarco, Tom; Controlling Software Projects; Yourdon Press, New York; 1982; ISBN 0-917072-32-4; 284 pages.

DeMarco, Tom and Lister, Tim; Peopleware; Dorset House Press, New York, NY; 1987; ISBN 0-932633-05-6; 188 pages.

DeMarco, Tom; Why Does Software Cost So Much?; Dorset House Press, New York, NY; ISBN 0-932633-34-X; 1995; 237 pages.

DeMarco, Tom; Deadline; Dorset House Press, New York, NY; 1997.

Department of the Air Force; Guidelines for Successful Acquisition and Management of Software Intensive Systems; Volumes 1 and 2; Software Technology Support Center, Hill Air Force Base, UT; 1994.

Dreger, Brian; Function Point Analysis; Prentice Hall, Englewood Cliffs, NJ; 1989; ISBN 0-13-332321-8; 185 pages.

Galea, R.B.; The Boeing Company: 3D Function Point Extensions, V2.0, Release 1.0; Boeing Information Support Services, Seattle, WA; June 1995.

Garmus, David & Herron, David; Measuring the Software Process: A Practical Guide to Functional Measurement; Prentice Hall, Englewood Cliffs, NJ; 1995.

Garmus, David & Herron, David; Function Point Analysis; Addison Wesley Longman, Boston, MA; 1996.

Garmus, David; Accurate Estimation; *Software Development*; July 1996; pp 57-65.

Grady, Robert B.; Practical Software Metrics for Project Management and Process Improvement; Prentice Hall, Englewood Cliffs, NJ; ISBN 0-13-720384-5; 1992; 270 pages.

Grady, Robert B. & Caswell, Deborah L.; Software Metrics: Establishing a Company-Wide Program; Prentice Hall, Englewood Cliffs, NJ; ISBN 0-13-821844-7; 1987; 288 pages.

Gulledge, Thomas R., Hutzler, William P.; and Lovelace, Joan S.(Editors); Cost Estimating and Analysis - Balancing Technology with Declining Budgets; Springer-Verlag; New York; ISBN 0-387-97838-0; 1992; 297 pages.

Howard, Alan (Ed.); Software Metrics and Project Management Tools; Applied Computer Research (ACR; Phoenix, AZ; 1997; 30 pages.

Humphrey, Watts S.; Managing the Software Process; Addison Wesley Longman, Reading, MA; 1989.

Humphrey, Watts; Personal Software Process; Addison Wesley Longman, Reading, MA; 1997.

IFPUG Counting Practices Manual, Release 4, International Function Point Users Group, Westerville, OH; April 1995; 83 pages.

Jones, Capers; "Measuring Programming Quality and Productivity; IBM Systems Journal; Vol. 17, No. 1; 1978; pp. 39-63.

Jones, Capers; Programming Productivity - Issues for the Eighties; IEEE Computer Society Press, Los Alamitos, CA; First edition 1981; Second edition 1986; ISBN 0-8186—0681-9; IEEE Computer Society Catalog 681; 489 pages.

Jones, Capers; SPQR/20 Users Guide; Software Productivity Research, Cambridge, MA; 1986; 55 pages.

Jones, Capers; Programming Productivity; McGraw Hill, New York, NY; ISBN 0-07-032811-0; 1986; 280 pages.

Jones, Capers; "A Ten-Year Retrospective of the ITT Programming Technology Center"; Software Productivity Research, Burlington, MA; 1988.

Jones, Capers; Applied Software Measurement; McGraw Hill, 2nd edition 1996; ISBN 0-07-032826-9; 618 pages.

Jones, Capers; Critical Problems in Software Measurement; Information Systems Management Group, 1993; ISBN 1-56909-000-9; 195 pages.

Jones, Capers; Software Productivity and Quality Today -- The Worldwide Perspective; Information Systems Management Group, 1993; ISBN -156909-001-7; 200 pages.

Jones, Capers; Assessment and Control of Software Risks; Prentice Hall, 1994; ISBN 0-13-741406-4; 711 pages.

Jones, Capers; New Directions in Software Management; Information Systems Management Group; ISBN 1-56909-009-2; 150 pages.

Jones, Capers; Patterns of Software System Failure and Success; International Thomson Computer Press, Boston, MA; December 1995; 250 pages; ISBN 1-850-32804-8; 292 pages.

Jones, Capers; Table of Programming Languages and Levels (8 Versions from 1985 through July 1996); 67 pages for Version 8; Software Productivity Research, Inc., Burlington, MA; 1996.

Jones, Capers; The Year 2000 Software Problem - Quantifying the Costs and Assessing the Consequences; Addison Wesley, Reading, MA; 1998; ISBN 0-201-30964-5; 303 pages.

Jones, Capers; Software Quality – Analysis and Guidelines for Success; International Thomson Computer Press, Boston, MA; ISBN 1-85032-876-6; 1997; 492 pages.

Jones, Capers; Estimating Software Costs; McGraw Hill, New York; 1998; 724 pages; ISBN 0-07-913094-1.

Jones, Capers; "The Economics of Object-Oriented Software"; SPR Technical Report; Software Productivity Research, Burlington, MA; April 1997; 22 pages.

Jones, Capers; "Becoming Best in Class"; SPR Technical Report; Software Productivity Research, Burlington, MA; January 1998; 40 pages.

Jones, Capers; "The Costs, Schedule, and Value of Software Process Improvement"; SPR Technical Report; Software Productivity Research, Inc.; Burlington, MA, January 1998; 27 pages.

Jones, Capers: "Revitalizing Project Management"; SPR Technical Report; Software Productivity Research, Inc.; Burlington, MA; August 1997; 37 pages.

Jones, Capers; "Estimating Rules of Thumb for the Year 2000 and Euro-Currency Projects"; SPR Technical Report; Software Productivity Research, Inc.; Burlington, MA; January 1998.

Jones, Capers; "Software Project Management Practices: Failure Versus Success"; Crosstalk, October 2004.

Jones, Capers; "Software Estimating Methods for Large Projects"; Crosstalk, April 2005.

Kan, Stephen H.; Metrics and Models in Software Quality Engineering, 2nd edition; Addison Wesley Longman, Boston, MA; ISBN 0-201-72915-6; 2003; 528 pages.

Kemerer, Chris F.; "An Empirical Validation of Software Cost Estimation Models; Communications of the ACM; 30; May 1987; pp. 416-429.

Kemerer, C.F.; "Reliability of Function Point Measurement - A Field Experiment"; Communications of the ACM; Vol. 36; pp 85-97; 1993.

Keys, Jessica; Software Engineering Productivity Handbook; McGraw Hill, New York, NY; ISBN 0-07-911366-4; 1993; 651 pages.

Love, Tom; Object Lessons; SIGS Books, New York; ISBN 0-9627477 3-4; 1993; 266 pages.

Marciniak, John J. (Editor); Encyclopedia of Software Engineering; John Wiley & Sons, New York; 1994; ISBN 0-471-54002; in two volumes.

McCabe, Thomas J.; "A Complexity Measure"; IEEE Transactions on Software Engineering; December 1976; pp. 308-320.

Melton, Austin; Software Measurement; International Thomson Press, London, UK; ISBN 1-85032-7178-7; 1995.

Mertes, Karen R.; Calibration of the CHECKPOINT Model to the Space and Missile Systems Center (SMC) Software Database (SWDB); Thesis AFIT/GCA/LAS/96S-11, Air Force Institute of Technology (AFIT), Wright Patterson AFB, Ohio; September 1996; 119 pages.

Mills, Harlan; Software Productivity; Dorset House Press, New York, NY; ISBN 0-932633-10-2; 1988; 288 pages.

Muller, Monika & Abram, Alain (editors); Metrics in Software Evolution; R. Oldenbourg Vertag GmbH, Munich; ISBN 3-486-23589-3; 1995.

Multiple authors; Rethinking the Software Process; (CD-ROM); Miller Freeman, Lawrence, KS; 1996. (This is a new CD ROM book collection jointly produced by the book publisher, Prentice Hall, and the journal publisher, Miller Freeman. This CD ROM disk contains the full text and illustrations of five Prentice Hall books: Assessment and Control of Software Risks by Capers Jones; Controlling Software Projects by Tom DeMarco; Function Point Analysis by Brian Dreger; Measures for Excellence by Larry Putnam and Ware Myers; and Object-Oriented Software Metrics by Mark Lorenz and Jeff Kidd.)

Park, Robert E. et al; Software Cost and Schedule Estimating - A Process Improvement Initiative; Technical Report CMU/SEI 94-SR-03; Software Engineering Institute, Pittsburgh, PA; May 1994.

Park, Robert E. et al; Checklists and Criteria for Evaluating the Costs and Schedule Estimating Capabilities of Software Organizations; Technical Report CMU/SEI 95-SR-005; Software Engineering Institute, Pittsburgh, PA; January 1995.

Paulk Mark et al; The Capability Maturity Model; Guidelines for Improving the Software Process; Addison Wesley, Reading, MA; ISBN 0-201-54664-7; 1995; 439 pages.

Perlis, Alan J., Sayward, Frederick G., and Shaw, Mary (Editors); Software Metrics; The MIT Press, Cambridge, MA; ISBN 0-262-16083-8; 1981; 404 pages.

Perry, William E.; Data Processing Budgets - How to Develop and Use Budgets Effectively; Prentice Hall, Englewood Cliffs, NJ; ISBN 0-13-196874-2; 1985; 224 pages.

Perry, William E.; Handbook of Diagnosing and Solving Computer Problems; TAB Books, Inc.; Blue Ridge Summit, PA; 1989; ISBN 0-8306-9233-9; 255 pages.

Pressman, Roger; Software Engineering - A Practitioner's Approach; McGraw Hill, New York, NY; 1982.

Putnam, Lawrence H.; Measures for Excellence -- Reliable Software On Time, Within Budget; Yourdon Press - Prentice Hall, Englewood Cliffs, NJ; ISBN 0-13-567694-0; 1992; 336 pages.

Putnam, Lawrence H and Myers, Ware.; Industrial Strength Software - Effective Management Using Measurement; IEEE Press, Los Alamitos, CA; ISBN 0-8186-7532-2; 1997; 320 pages.

Reifer, Donald (editor); Software Management (4th edition); IEEE Press, Los Alamitos, CA; ISBN 0 8186-3342-6; 1993; 664 pages.

Roetzheim, William H. and Beasley, Reyna A.; Best Practices in Software Cost and Schedule Estimation; Prentice Hall PTR, Saddle River, NJ; 1998.

Rubin, Howard; Software Benchmark Studies For 1997; Howard Rubin Associates, Pound Ridge, NY; 1997.

Shepperd, M.: "A Critique of Cyclomatic Complexity as a Software Metric"; Software Engineering Journal, Vol. 3, 1988; pp. 30-36.

Software Productivity Consortium; The Software Measurement Guidebook; International Thomson Computer Press; Boston, MA; ISBN 1-850-32195-7; 1995; 308 pages.

St-Pierre, Denis; Maya, Marcela; Abran, Alain, and Desharnais, Jean-Marc; Full Function Points: Function Point Extensions for Real-Time Software, Concepts and Definitions; University of Quebec. Software Engineering Laboratory in Applied Metrics (SELAM); TR 1997-03; March 1997; 18 pages.

Strassmann, Paul; The Squandered Computer; The Information Economics Press, New Canaan, CT; ISBN 0-9620413-1-9; 1997; 426 pages.

Stukes, Sherry, Deshoretz, Jason, Apgar, Henry and Macias, Ilona; Air Force Cost Analysis Agency Software Estimating Model Analysis ; TR-9545/008-2; Contract F04701-95-D-0003, Task 008; Management Consulting & Research, Inc.; Thousand Oaks, CA 91362; September 30 1996.

Symons, Charles R.; Software Sizing and Estimating – Mk II FPA (Function Point Analysis); John Wiley & Sons, Chichester; ISBN 0 471-92985-9; 1991; 200 pages.

Thayer, Richard H. (editor); Software Engineering and Project Management; IEEE Press, Los Alamitos, CA; ISBN 0 8186-075107; 1988; 512 pages.

Umbaugh, Robert E. (Editor); Handbook of IS Management; (Fourth Edition); Auerbach Publications, Boston, MA; ISBN 0-7913-2159-2; 1995; 703 pages.

Whitmire, S.A.; "3-D Function Points: Scientific and Real-Time Extensions to Function Points"; Proceedings of the 1992 Pacific Northwest Software Quality Conference, June 1, 1992.

Yourdon, Ed; Death March - The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects; Prentice Hall PTR, Upper Saddle River, NJ; ISBN 0-13-748310-4; 1997; 218 pages.

Zells, Lois; Managing Software Projects - Selecting and Using PC-Based Project Management Systems; QED Information Sciences, Wellesley, MA; ISBN 0-89435-275-X; 1990; 487 pages.

Zuse, Horst; Software Complexity - Measures and Methods; Walter de Gruyter, Berlin; 1990; ISBN 3-11-012226-X; 603 pages.

Zuse, Horst; A Framework of Software Measurement; Walter de Gruyter, Berlin; 1997.