

Introduction to Software Cost Estimation

Software cost estimation is a complex activity that requires knowledge of a number of key attributes about the project for which the estimate is being constructed. Cost estimating is sometimes termed “parametric estimating” because accuracy demands understanding the relationships among scores of discrete parameters that can affect the outcomes of software projects, both individually and in concert. Creating accurate software cost estimates requires knowledge of the following parameters:

- *The sizes of major deliverables, such as specifications, source code, and manuals*
- *The rate at which requirements are likely to change during development*
- *The probable number of bugs or defects that are likely to be encountered*
- *The capabilities of the development team*
- *The salaries and overhead costs associated with the development team*
- *The formal methodologies that are going to be utilized (such as the Agile methods)*
- *The tools that are going to be utilized on the project*

2 Section 1: Introduction to Software Cost Estimation

- *The set of development activities that are going to be carried out*
- *The cost and schedule constraints set by clients of the project being estimated*

Although the factors that influence the outcomes of software projects are numerous and some are complex, modern commercial software cost-estimation tools can ease the burden of project managers by providing default values for all of the key parameters, using industry values derived from the integral knowledge base supplied with the estimation tools.

In addition, software cost-estimation tools allow the construction of customized estimating templates that are derived from actual projects and that can be utilized for estimating projects of similar sizes and kinds.

This section discusses the origins and evolution of software cost-estimation tools and how software cost estimation fits within the broader category of software project management. In addition, this section discusses the impact of software cost-estimation tools on the success rates of software projects and uses this data to illustrate the approximate return on investment (ROI) from software cost-estimating and project management tools.

Introduction

Software cost estimating has been an important but difficult task since the beginning of the computer era in the 1940s. As software applications have grown in size and importance, the need for accuracy in software cost estimating has grown, too.

In the early days of software, computer programs were typically less than 1000 machine instructions in size (or less than 30 function points), required only one programmer to write, and seldom took more than a month to complete. The entire development costs were often less than \$5000. Although cost estimating was difficult, the economic consequences of cost-estimating errors were not very serious.

Today some large software systems exceed 25 million source code statements (or more than 300,000 function points), may require technical staffs of 1000 personnel or more, and may take more than five calendar years to complete. The development costs for such large software systems can exceed \$500 million; therefore, errors in cost estimation can be very serious indeed.

Even more serious, a significant percentage of large software systems run late, exceed their budgets, or are canceled outright due to severe underestimating during the requirements phase. In fact, excessive optimism in software cost estimation is a major source of overruns, failures, and litigation.

Software is now the driving force of modern business, government, and military operations. This means that a typical Fortune 500 corporation or a state government may produce hundreds of new applications and modify hundreds of existing applications every year. As a result of the host of software projects in the modern world, software cost estimating is now a mainstream activity for every company that builds software.

In addition to the need for accurate software cost estimates for day-to-day business operations, software cost estimates are becoming a

significant aspect in litigation. Over the past fifteen years, the author and his colleagues have observed dozens of lawsuits where software cost estimates were produced by the plaintiffs, the defendants, or both. For example, software cost estimation now plays a key part in lawsuits involving the following disputes:

- Breach of contract suits between clients and contractors
- Suits involving the taxable value of software assets
- Suits involving recovering excess costs for defense software due to scope expansion
- Suits involving favoritism in issuance of software contracts
- Suits involving wrongful termination of software personnel

From many viewpoints, software cost estimating has become a critical technology of the 21st century because software is now so pervasive.

How Software Cost-Estimating Tools Work

There are many kinds of automated tools that experienced project managers can use to create cost, schedule, and resource estimates for software projects. For example, an experienced software project manager can create a cost-estimate and schedule plan using any of the following:

- Spreadsheets
- Project management tools
- Software cost-estimating tools

A frequently asked question for software cost-estimating tool vendors is “Why do we need your tool when we already have spreadsheets and project management tools?”

The commercial software-estimating tools are differentiated from all other kinds of software project management tools and general-purpose tools, such as spreadsheets, in these key attributes:

- They contain knowledge bases of hundreds or thousands of software projects.
- They can perform size predictions, which general-purpose tools cannot.
- They can automatically adjust estimates based on tools, languages, and processes.
- They can predict quality and reliability, which general-purpose tools cannot.

- They can predict maintenance and support costs after deployment.
- They can predict (and prevent) problems long before the problems actually occur.

Unlike other kinds of project management tools, the commercial software cost-estimating tools do not depend upon the expertise of the user or project manager, although experienced managers can refine the estimates produced. The commercial cost-estimating tools contain the accumulated experience of many hundreds or thousands of software projects.

Because of the attached knowledge bases associated with commercial cost-estimating tools, novice managers or managers faced with unfamiliar kinds of projects can describe the kind of project being dealt with, and the estimating tool will construct an estimate based on similar projects derived from information contained in its associated knowledge base.

Figure 1.1 illustrates the basic principles of modern commercial software cost-estimating tools.

The starting point of software estimation is the size of the project in terms of either logical source code statements, physical lines of code, function points, or, sometimes, all three metrics. The project's size can be derived from the estimating tool's own sizing logic, supplied by users as an explicit input, or derived from analogy with similar projects stored in the estimating tool's knowledge base. Even for Agile projects and those using iterative development, at least approximate size information can be created.

Once the basic size of the project has been determined, the estimate can be produced based on the specific attributes of the project in question. Examples of the attributes that can affect the outcome of the estimate include the following:

- The rate at which project requirements may change
- The experience of the development team with this kind of project
- The process or methods used to develop the project ranging from Agile to Waterfall

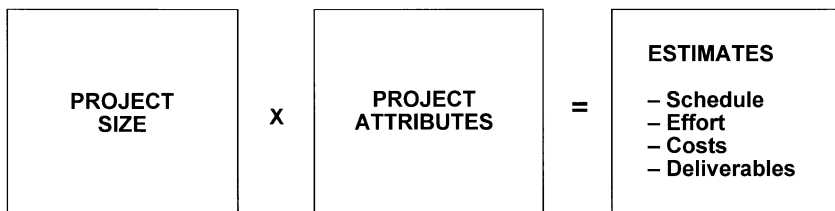


Figure 1.1 Software-estimating principles.

- The specific activities that will be performed during development
- The number of increments, iterations, or “sprints” that will be used
- The programming language or languages utilized
- The presence or absence of reusable artifacts
- The development tool suites used to develop the project
- The environment or ergonomics of the office work space
- The geographic separation of the team across multiple locations
- The schedule pressure put on the team by clients or executives
- Contractual obligations in terms of costs, dates, defects, or features

Using commercial estimating tools, these project attributes can either be supplied by the user or inherited from similar projects already stored in the estimating tool’s knowledge base. In a sense estimating tools share some of the characteristics of the object-oriented paradigm in that they allow inheritance of shared attributes from project to project.

In software-estimating terminology, these shared attributes are termed *templates*, and they can be built in a number of ways. For example, estimating-tool users can point to an existing completed project and select any or all of the features of that project as the basis of the template. Thus, if the project selected as the basis of a template were a systems software project, used the C programming language, and utilized formal design and code inspections, these attributes could be inherited as part of the development cycle and could become part of an estimating template for other projects.

Many other attributes from historical projects can also be inherited and can become aspects of software-estimating templates. For example, a full estimating template can contain inherited attribute data on such topics as the following:

- The experience of the development team in similar applications
- The process or methodology used to develop the application
- The SEI capability maturity level of the organization
- The standards that will be adhered to, such as ISO, DoD, IEEE, and so forth
- The tools used during design, coding, testing, and so forth
- The programming language or languages utilized
- The volumes of reusable artifacts available
- The ergonomics of the programming office environment

Since software projects are not identical, any of these inherited attributes can be modified as the need arises. However, the availability of

templates makes the estimation process quicker, more convenient, and more reliable because templates substitute specific knowledge from local projects for generic industry default values.

Templates can also be derived from sets of projects rather than from one specific project, or can even be custom-built by the users, using artificial factors. However, the most common method of template development is to use the automatic template construction ability of the estimating tool, and simply select relevant historical projects to be used as the basis for the template.

As a general rule, software-estimating templates are concerned with four key kinds of inherited attributes: (1) *personnel*, (2) *technologies*, (3) *tools*, and (4) the *programming environment*, as illustrated by Figure 1.2.

Three of these four factors—the experience of the personnel, the development process, and the technology (programming languages and support tools)—are fairly obvious in their impact. What is not obvious, but is equally important, is the impact of the fourth factor—*environment*.

The environment factor covers individual office space and the communication channels among geographically dispersed development teams. Surprisingly, access to a quiet, noise-free office environment is one of the major factors that influences programming productivity.

The ability to include ergonomic factors in an estimate is an excellent example of the value of commercial software cost-estimating tools. Not only do they contain the results of hundreds or thousands of completed projects, but the tools contain data about influential factors that many human project managers may not fully understand.

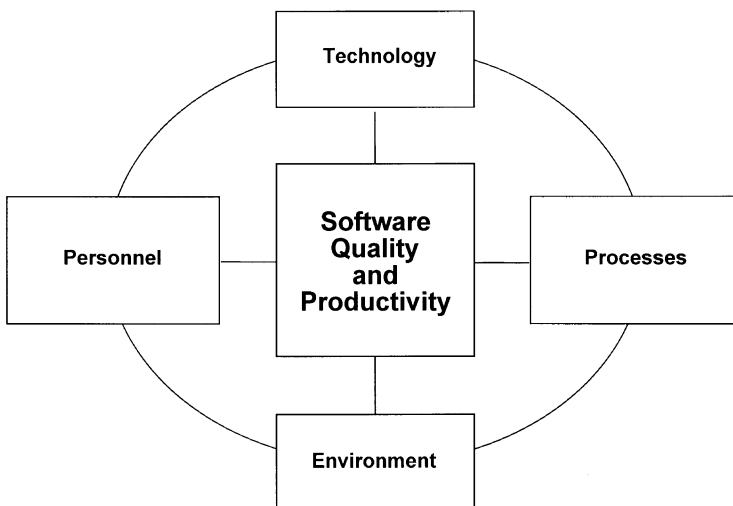


Figure 1.2 Key estimate factors.

The four key sets of attributes must be considered whether estimating manually or using an automated estimating tool. However, one of the key features of commercial software-estimating tools is the fact that they are repositories containing the results of hundreds or thousands of software projects, and so the effect of these four attribute areas can be examined, and their impacts can be analyzed.

There is a standard sequence for software cost estimation, which the author has used for more than 35 years. This sequence can be used with manual software cost estimates, and also mirrors the estimation stages in the software-estimation tools that the author has designed. There are ten steps in this sequence, although the sequence starts with 0 because the first stage is a pre-estimate analysis of the requirements of the application.

Step 0: Analyze the Requirements

Software cost estimation at the project level cannot be performed unless the requirements of the project are well understood. Therefore, before estimating itself can begin, it is necessary to explore and understand the user requirements. At some point in the future it should be possible to create estimates automatically from the requirements specifications, but the current level of estimating technology demands human intervention.

A common estimating activity today is to analyze the software requirements and create function point totals based on those requirements. This provides the basic size data used for formal cost estimation. Function point analysis is usually performed manually by certified function point counting personnel. A time is rapidly approaching when function point totals can be derived automatically from software requirements, and this method may appear in commercial software cost-estimation tools within a few years.

It is a known fact that requirements for large systems cannot be fully defined at the start of the project. This fact is the basis for the Agile methods, extreme programming, Scrum, and a number of others. This fact is also embedded in the algorithms for several commercial software estimating tools. Once the initial requirements are understood, the average rate of growth of new requirements is about 2 percent per calendar month. This growth can be planned for and included in the estimate.

Step 1: Start with Sizing

Every form of estimation and every commercial software cost-estimating tool needs the sizes of key deliverables in order to complete an estimate. Size data can be derived in several fashions, including the following:

- Size prediction using an estimating tool's built-in sizing algorithms
- Sizing by extrapolation from function point totals
- Sizing by analogy with similar projects of known size
- Guessing at the size using "project manager's intuition"
- Guessing at the size using "programmer's intuition"
- Sizing using statistical methods or Monte Carlo simulation

For Agile methods and those projects using iterative development, sizing of the entire application may be deferred until the early increments are complete. However, even for Agile and iterative projects it is possible to make an approximate prediction of final size just by comparing the nature of the project to similar projects or using size approximations based on the class, type, and nature of the software. Later in this book examples are given of such sizing methods.

The basic size of the application being estimated is usually expressed in terms of function points, source code statements, or both. However, it is very important to size all deliverables and not deal only with code. For example, more than 50 kinds of paper documents are associated with large software projects, and they need to be sized also. Of course, when using one of the commercial software estimating tools that support documents sizing, these sizes will automatically be predicted.

Source code sizing must be tailored to specific programming languages, and more than 600 languages are now in use. About one-third of software projects utilize more than a single programming language. More than a dozen kinds of testing occur, and each will require different volumes of test cases.

Sizing is a key estimating activity. If the sizes of major deliverables can be predicted within 5 to 10 percent, then the accuracy of the overall estimate can be quite good. If size predictions are wildly inaccurate, then the rest of the estimate will be inaccurate, too. As mentioned earlier, empirical evidence from large software projects indicates that requirements grow at an average rate of about 2 percent per calendar month from the end of the requirements phase until the start of the testing phase. The total growth of requirements can exceed 50 percent of the volume of the initial requirements when measured with function points. A major problem with estimating accuracy has been ignoring or leaving out requirements creep. Modern cost-estimating tools can predict requirements growth and can include their costs and schedules in the estimate.

The technologies available for sizing have been improving rapidly. In the early days of software cost estimation, size data had to be supplied by users, using very primitive methods. Now modern software

cost-estimating tools have a number of sizing capabilities available, including support for very early size estimates even before the requirements are firm.

Step 2: Identify the Activities to Be Included

Once the sizes of key deliverables are known, the next step is to select the set of activities that are going to be performed. In this context the term *activities* refers to the work that will be performed for the project being estimated, such as requirements, internal design, external design, design inspections, coding, code inspections, user document creation, meetings or Scrum sessions, change control, integration, quality assurance, unit testing, new function testing, regression testing, system testing, and project management. Accurate estimation is impossible without knowledge of the activities that are going to be utilized.

Activity patterns vary widely from project to project. Large systems utilize many more activities than do small projects. Waterfall projects utilize more activities than Agile projects. For projects of the same size, military and systems software utilize more activities than do information systems. Local patterns of activities are the ones to utilize, because they reflect your own enterprise's software development methodologies.

Modern software cost-estimating tools have built-in logic for selecting the activity patterns associated with many kinds of software development projects. Users can also adjust the activity patterns to match local variations.

Step 3: Estimate Software Defect Potentials and Removal Methods

The most expensive and time-consuming work of software development is the work of finding bugs and fixing them. In the United States the number of bugs or defects in requirements, design, code, documents, and bad-fixes averages five per function point. Average defect removal efficiency before delivery is 85 percent. The cost of finding and repairing these defects averages about 35 percent of the total development cost of building the application. The schedule time required is about 30 percent of project development schedules. Defect repair costs and schedules are often larger than coding costs and schedules. Accuracy in software cost estimates is not possible if defects and defect removal are not included in the estimates. The author's software cost-estimating tools include full defect-estimation capabilities, and support all known kinds of defect-removal activity. This is necessary because the total effort, time, and cost devoted to a full series of reviews, inspections, and multistage tests will cost far more than source code itself.

Defect estimation includes predictive abilities for requirements defects, design defects, coding defects, user documentation defects, and a very troubling category called *bad fix defects*. The phrase *bad fix* refers to new defects accidentally injected as a by-product of repairing previous defects. Bad fix defects average about 7 percent of all defect repairs. Some estimating tools can predict bad fixes.

Estimating tools that support commercial software can also predict duplicate defect reports, or bugs found by more than one customer. It is also possible to estimate *invalid defect reports*, or bug reports that turn out not to be the fault of the software, such as user errors or hardware problems.

The ability to predict software defects would not be very useful without another kind of estimation, which is predicting the *defect-removal efficiency* of various kinds of reviews, inspections, and test stages. Modern software cost-estimating tools can predict how many bugs will be found by every form of defect removal, from desk checking through external beta testing.

Step 4: Estimate Staffing Requirements

Every software deliverable has a characteristic *assignment scope*, or amount of work that can be done by a single employee. For example, an average assignment for an individual programmer will range from 5000 to 15,000 source code statements (from about 50 up to 2000 function points).

However, large systems also utilize many specialists, such as system architects, database administrators, quality assurance specialists, software engineers, technical writers, testers, and the like. Identifying each category of worker and the numbers of workers for the overall project is the next step in software cost estimation.

Staffing requirements depend upon the activities that will be performed and the deliverables that will be created, so staffing predictions are derived from knowledge of the overall size of the application and the activity sets that will be included. Staffing predictions also need to be aware of “pair programming” or two-person teams, which are part of some of the new Agile methods.

For large systems, programmers themselves may comprise less than half of the workforce. Various kinds of specialists and project managers comprise the other half. Some of these specialists include quality assurance personnel, testing personnel, technical writers, systems analysts, database administrators, and configuration control specialists. If the project is big enough to need specialists, accurate estimation requires that their efforts be included. Both programming and other kinds of noncoding activities,

such as production of manuals and quality assurance, must be included to complete the estimate successfully.

Step 5: Adjust Assumptions Based on Capabilities and Experience

Software personnel can range from top experts with years of experience to rank novices on their first assignment. Once the categories of technical workers have been identified, the next step is to make adjustments based on typical experience levels and skill factors.

Experts can take on more work, and perform it faster, than can novices. This means that experts will have larger assignment scopes and higher production rates than average or inexperienced personnel.

Other adjustments include work hours per day, vacations and holidays, unpaid and paid overtime, and assumptions about the geographic dispersal of the software team. Adjusting the estimate to match the capabilities of the team is one of the more critical estimating activities.

While estimating tools can make adjustments to match varying degrees of expertise, these tools have no way of knowing the specific capabilities of any given team. Many commercial estimating tools default to “average” capabilities, and allow users to adjust this assumption upward or downward to match specific team characteristics.

Step 6: Estimate Effort and Schedules

Effort and schedule estimates are closely coupled, and often are performed in an iterative manner.

Accurate effort estimation requires knowledge of the basic size of the application plus the numbers and experience levels of the software team members and the sizes of various deliverables they are expected to produce, such as specifications and user manuals.

Accurate schedule estimation requires knowledge of the activities that will be performed, the number of increments or “sprints” that will be carried out, the sizes of various deliverables, the overlap between activities with mutual dependencies, and the numbers and experience levels of the software team members.

Schedule and effort estimates are closely coupled, but the interaction between these two dimensions is complicated and sometimes is counterintuitive. For example, if a software project will take six months if it is developed by one programmer, adding a second programmer will not cut the schedule to three months. Indeed, a point can be reached where putting on additional personnel may slow down the project’s schedule rather than accelerating it.

The complex sets of rules that link effort and schedules for software projects are the heart of the algorithms for software cost-estimating tools.

As an example of one of the more subtle rules that estimating tools contain, adding personnel to a software project within one department will usually shorten development schedules. But if enough personnel are added so that a second department is involved, schedules will stretch out. The reason for this is that software schedules, and also productivity rates, are inversely related to the number of project managers engaged. There are scores of rules associated with the interaction of schedules and effort, and some of these are both subtle and counterintuitive.

In fact, for very large software projects with multiple teams, the rate at which development productivity declines tends to correlate more closely to the number of managers that are engaged than to the actual number of programmers involved. This phenomenon leads to some subtle findings, such as the fact that projects with a small span of control (less than six developers per manager) may have lower productivity than similar projects with a large span of control (12 developers per manager).

Step 7: Estimate Development Costs

Development costs are the next-to-last stage of estimation and are very complex. Development costs are obviously dependent upon the effort and schedule for software projects, so these factors are predicted first, and then costs are applied afterwards.

Costs for software projects that take exactly the same amount of effort in terms of hours or months can vary widely due to the following causes:

- Average salaries of workers and managers on the project
- The corporate burden rate or overhead rate applied to the project
- Inflation rates, if the project will run for several years
- Currency exchange rates, if the project is developed internationally

There may also be special cost topics that will have to be dealt with separately, outside of the basic estimate:

- License fees for any acquired software needed
- Capital costs for any new equipment
- Moving and living costs for new staff members
- Travel costs for international projects or projects developed in different locations
- Contractor and subcontractor costs
- Legal fees for copyrights, patents, or other matters
- Marketing and advertising costs

- Costs for developing videos or CD-ROM tutorial materials and training
- Content acquisition costs if the application is a web-based product

On the whole, developing a full and complete cost estimate for a software project is much more complex than simply developing a resource estimate of the number of work hours that are likely to be needed. Many cost elements, such as burden rates or travel, are only indirectly related to effort and can impact the final cost of the project significantly.

The normal pattern of software estimation is to use hours, days, weeks, or months of effort as the primary estimating unit, and then apply costs at the end of the estimating cycle once the effort patterns have been determined.

Step 8: Estimate Maintenance and Enhancement Costs

Software projects often continue to be used and modified for many years. Maintenance and enhancement cost estimation are special topics, and are more complex than new project cost estimation.

Estimating maintenance costs requires knowledge of the probable number of users of the application, combined with knowledge of the probable number of bugs or defects in the product at the time of release.

Estimating enhancement costs requires good historical data on the rate of change of similar projects once they enter production and start being used. For example, new software projects can add 10 percent or more in total volume of new features with each release for several releases in a row, but then slow down for a period of two to three years before another major release occurs.

Many commercial estimating tools can estimate both the initial construction costs of a project and the maintenance and enhancement cost patterns for more than five years of usage by customers.

There is no actual limit on the number of years that can be estimated, but because long-range projections of user numbers and possible new features are highly questionable, the useful life of maintenance and enhancement estimates runs from three to five years. Estimating maintenance costs ten years into the future can be done, but no estimate of that range can be regarded as reliable because far too many uncontrollable business variables can occur.

Step 9: Present Your Estimate to the Client and Defend It Against Rejection

Once a cost estimate is prepared, the next step is to present the estimate to the client who is going to fund the project. For large systems and

applications of 5000 function points or larger (equivalent to roughly 500,000 source code statements) about 60 percent of the initial estimates will be rejected by the client. Either the estimated schedule will be too long, the costs will be too high, or both. Often the client will decree a specific delivery date much shorter than the estimated date. Often the client will decree that costs must be held within limits much lower than the estimated costs. Projects where formal estimates are rejected and replaced by arbitrary schedules and costs derived from external business needs rather than team capabilities have the highest failure rates in the industry. About 60 percent of such projects will be cancelled and never completed at all. (At the point of cancellation, both costs and schedules will already have exceeded their targets.) Of the 40 percent of projects that finally do get completed, the average schedule will be about one year late, and the average cost will be about 50 percent higher than targets.

The best defense against having a cost estimate rejected is to have solid historical data from at least a dozen similar projects. The second-best defense against have a cost estimate rejected is to prepare a full activity-based estimate that includes quality, paperwork, requirements creep, all development activities, and all maintenance tasks, You will need to prove that your estimate has been carefully prepared and has left nothing to chance. High-level or phase-level estimates that lack detail are not convincing and are easy to reject.

Cautions About Accidental Omissions from Estimates

Because software-estimating tools have such an extensive knowledge base, they are not likely to make the kinds of mistakes that inexperienced human managers make when they create estimates by hand or with general-purpose tools and accidentally omit activities from the estimate.

For example, when estimating large systems, coding may be only the fourth most expensive activity. Human managers often tend to leave out or underestimate the non-code work, but estimating tools can include these other activities.

- Historically, the effort devoted to finding and fixing bugs by means of reviews, walkthroughs, inspections, and testing takes more time and costs more than any other software activities. Therefore, accurate cost estimates need to start with quality predictions, because defect-removal costs are often more expensive than anything else.
- In second place as major cost elements are the expenses and effort devoted to the production of paper documents, such as plans, specifications, user manuals, and the like. For military software projects,

paperwork will cost twice as much as the code itself. For large civilian projects greater than 1000 function points or 100,000 source code statements, paper documents will be a major cost element and will approach or exceed the cost of the code.

- In third place for many large projects are the costs and schedules of dealing with “creeping requirements” or new features added to the project after the requirements phase. All software projects will grow due to creeping requirements and therefore this factor should be an integral part of the estimates for all major software projects.
- For some large distributed applications that involve multiple development locations or subcontractors, the costs of meetings and travel among the locations can cost more than the source code and may be in fourth place in the sequence of all software costs. A frequent omission from software cost estimates is the accidental exclusion of travel costs (airlines, hotels, etc.) for meetings among the development teams that are located in different cities or different countries. For very large kinds of systems, such as operating systems, telecommunication systems, or defense systems, which may involve distributed development in half a dozen countries and a dozen cities, the costs of travel can exceed the cost of coding significantly, and this topic should not be left out by accident.
- Many software cost estimates—and many measurement systems, too—cover only the core activities of software development and ignore such topics as project management and support (i.e., program librarians, secretaries, administration, etc.). These ancillary activities are part of the project and can, in some cases, top 20 percent of total costs. This is far too much to leave out by accident.
- The software domain has fragmented into a number of specialized skills and occupations. It is very common to accidentally leave out the contributions of specialists if their skills are needed only during portions of a software development cycle. Some of the specialist groups that tend to be accidentally omitted from software cost estimates include quality assurance specialists, technical writing specialists, function point specialists, database administration specialists, performance tuning specialists, network specialists, and system administration specialists. The combined contributions of these and other specialists may total more than 20 percent of all software development costs and should not be omitted by accident.
- The most common omission from internal software cost estimates for information systems are the costs expended by users during requirements definition, prototyping, status reviews, phase reviews, documentation, inspections, acceptance testing, and other activities where

the developers have a key role. Since user representatives are not usually considered to be part of the project team, their contributions to the project are seldom included in software cost estimates, and are seldom included in measurement studies, either. The actual amount of effort contributed by users to major software development projects can approach 20 percent of the total work in some cases, which is not a trivial amount and is far too significant to leave out by accident. Some commercial software cost-estimating tools keep a separate chart of accounts for user activities and allow user efforts to be added to total project costs, if desired.

- For many projects, maintenance after delivery quickly costs more than the development of the application itself. It is unwise to stop the estimate at the point of delivery of the software without including at least five years of maintenance and enhancement estimates. Since maintenance (defect repairs) and enhancements (adding new features) have different funding sources, many estimating tools separate these two activities. Other forms of maintenance work, such as customer support or field service, may also be included in post-release estimates.

A key factor that differentiates modern commercial software cost-estimating tools from general-purpose tools, such as spreadsheets and project management tools, is the presence of full life-cycle historical data. This gives them the ability to estimate quality and to estimate the sizes and costs of producing paper deliverables, the probable volumes of creeping requirements, and the costs of coding and testing.

When considering acquisition of a software cost-estimating tool, be sure that the knowledge base includes the kind of software you are interested in. The real-life cost and schedule results of information systems, systems software, commercial software, military software, and embedded software are not identical, and you need to be sure the estimating tool contains data on the kinds of software you are concerned with. Some tools support all classes of software, but others are more narrow in focus.

Software Cost Estimating and Other Development Activities

Software cost estimating is not a “standalone” activity. The estimates are derived in large part from the requirements of the project, and will be strongly affected by the tools, process, and other attributes associated with the project. A cost estimate is a precursor for departmental budgets, and also serves as a baseline document for comparing accumulated costs against projected costs.

For any project larger than trivial, multiple cost estimates will be prepared during the course of development, including but not limited to the following:

- A rough pre-requirements guesstimate
- An initial formal estimate derived from the project requirements
- One or more midlife estimates, which reflect requirements changes
- A final cost accumulation using project historical data

In addition, since the software industry is somewhat litigious, cost estimates may also be prepared as a by-product of several kinds of litigation, including the following:

- Litigation for breach of contract between software clients and out-source companies
- Litigation involving the taxable value of software in tax disputes

In the course of developing a software project, historical data will steadily be accumulated. This means that after the first rough guesstimate and the initial requirements estimate, future estimates will need to interleave historical data with predicted data. Therefore, software-estimating tools need the ability to capture historical data and to selectively display both historical data and predicted values.

Figure 1.3 illustrates how software cost estimation fits into the context of other key software development activities.

As can be seen from Figure 1.3, estimating is closely aligned with other key development phases. When done well, software cost estimates are among the most valuable documents in the entire software world, because they make a software project real and tangible in terms of the resources, schedules, and costs that will be required.

However, cost estimates that are poorly constructed and grossly inaccurate are key factors in almost every major software disaster. The best advice for those charged with constructing software cost estimates is the following:

- Be accurate.
- Be conservative.
- Base the estimate on solid historical data.
- Include quality, since software quality affects schedules and costs.
- Include paper documents, since they can cost more than source code.
- Include project management.

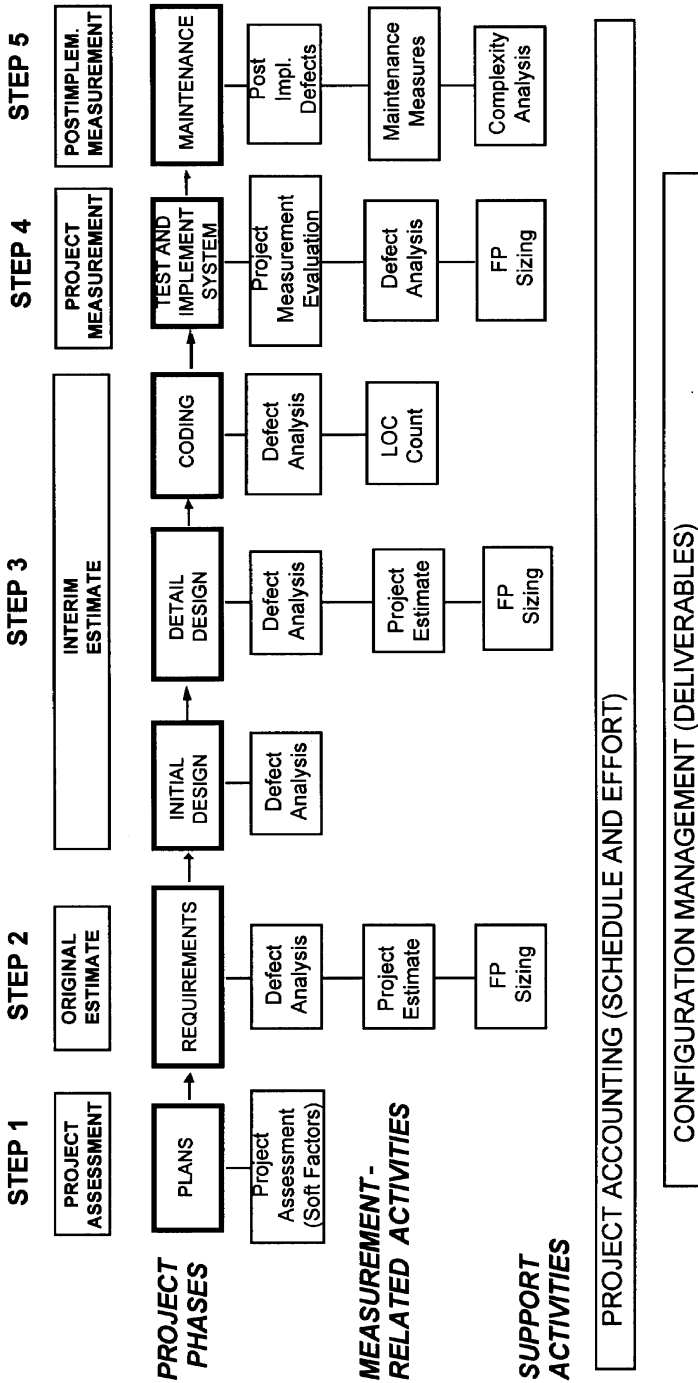


Figure 1.3 Software cost estimation and other activities.

- Include the effects of creeping requirements.
- Do not exaggerate the effect of tools, languages, or methods.
- Get below phases to activity-level cost estimates.
- Be prepared to defend the assumptions of your estimate.

Even with the best estimating tools, accurate software cost estimating is complicated and can be difficult. But without access to good historical data, accurate software cost estimating is almost impossible. Measurement and estimation are twin technologies and both are urgently needed by software project managers.

Measurement and estimation are also linked in the commercial software cost-estimation marketplace, since many of the commercial estimating companies are also benchmark and measurement companies. As better historical data becomes available, the features of the commercial software cost-estimating tools are growing stronger.

References

- Barrow, Dean, Susan Nilson, and Dawn Timberlake: *Software Estimation Technology Report*, Air Force Software Technology Support Center, Hill Air Force Base, Utah, 1993.
- Boehm, Barry: *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
- : *Software Cost Estimation with COCOMO II*, Prentice-Hall, Englewood Cliffs, NJ;
- Brown, Norm (ed.): *The Program Manager's Guide to Software Acquisition Best Practices*, Version 1.0, U.S. Department of Defense, Washington, D.C., July 1995.
- Charette, Robert N.: *Software Engineering Risk Analysis and Management*, McGraw-Hill, New York, 1989.
- : *Application Strategies for Risk Analysis*, McGraw-Hill, New York, 1990.
- Cohn, Mike: *Agile Estimating and Planning* (Robert C. Martin Series), Prentice-Hall PTR, Englewood Cliffs, NJ, 2005.
- Coombs, Paul: *IT Project Estimation: A Practical Guide to the Costing of Software*, Cambridge University Press, Melbourne, Australia.
- DeMarco, Tom: *Controlling Software Projects*, Yourdon Press, New York.
- : *Deadline*, Dorset House Press, New York, 1997.
- Department of the Air Force: *Guidelines for Successful Acquisition and Management of Software Intensive Systems*; vols. 1 and 2, Software Technology Support Center, Hill Air Force Base, Utah, 1994.
- Dreger, Brian: *Function Point Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1989.
- Galorath, Daniel D. and Michael W. Evans: *Software Sizing, Estimation, and Risk Management*, Auerbach, Philadelphia, PA, 2006.
- Garmus, David, and David Herron: *Measuring the Software Process: A Practical Guide to Functional Measurement*, Prentice-Hall, Englewood Cliffs, N.J., 1995.
- Garmus, David and David Herron: *Function Point Analysis: Measurement Practices for Successful Software Projects*, Addison-Wesley, Boston, Mass., 2001.
- Grady, Robert B.: *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
- and Deborah L. Caswell: *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Englewood Cliffs, N.J., 1987.

- Gulledge, Thomas R., William P. Hutzler, and Joan S. Lovelace (eds.): *Cost Estimating and Analysis—Balancing Technology with Declining Budgets*, Springer-Verlag, New York, 1992.
- Howard, Alan (ed.): *Software Metrics and Project Management Tools*, Applied Computer Research (ACR), Phoenix, Ariz., 1997.
- IFPUG Counting Practices Manual*, Release 4, International Function Point Users Group, Westerville, Ohio, April 1995.
- Jones, Capers: *Critical Problems in Software Measurement*, Information Systems Management Group, 1993a.
- : *Software Productivity and Quality Today—The Worldwide Perspective*, Information Systems Management Group, 1993b.
- : *Assessment and Control of Software Risks*, Prentice-Hall, Englewood Cliffs, N.J., 1994.
- : *New Directions in Software Management*, Information Systems Management Group.
- : *Patterns of Software System Failure and Success*, International Thomson Computer Press, Boston, 1995.
- : *Applied Software Measurement*, 2nd ed., McGraw-Hill, New York, 1996.
- : *The Economics of Object-Oriented Software*, Software Productivity Research, Burlington, Mass., April 1997a.
- : *Software Quality—Analysis and Guidelines for Success*, International Thomson Computer Press, Boston, 1997b.
- : *The Year 2000 Software Problem—Quantifying the Costs and Assessing the Consequences*, Addison-Wesley, Reading, Mass., 1998.
- : *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, Boston, Mass, 2000.
- Kan, Stephen H.: *Metrics and Models in Software Quality Engineering*, 2nd edition, Addison-Wesley, Boston, Mass., 2003.
- Kemerer, C. F.: “Reliability of Function Point Measurement—A Field Experiment,” *Communications of the ACM*, **36**: 85–97 (1993).
- Keys, Jessica: *Software Engineering Productivity Handbook*, McGraw-Hill, New York, 1993.
- Laird, Linda M. and Carol M. Brennan: *Software Measurement and Estimation: A practical Approach*; John Wiley & Sons, New York, 2006.
- Lewis, James P.: *Project Planning, Scheduling & Control*, McGraw-Hill, New York, New York, 2005.
- Marciniak, John J. (ed.): *Encyclopedia of Software Engineering*, vols. 1 and 2, John Wiley & Sons, New York, 1994.
- McConnell, Steve: *Software Estimation: Demystifying the Black Art*, Microsoft Press, Redmond, WA, 2006.
- 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.
- Ourada, Gerald, and Daniel V. Ferens: “Software Cost Estimating Models: A Calibration, Validation, and Comparison,” in *Cost Estimating and Analysis: Balancing Technology and Declining Budgets*, Springer-Verlag, New York, 1992, pp. 83–101.
- Perry, William E.: *Handbook of Diagnosing and Solving Computer Problems*, TAB Books, Blue Ridge Summit, Pa., 1989.
- Pressman, Roger: *Software Engineering: A Practitioner’s Approach with Bonus Chapter on Agile Development*, McGraw-Hill, New York, 2003.
- Putnam, Lawrence H.: *Measures for Excellence—Reliable Software on Time, Within Budget*: Yourdon Press/Prentice-Hall, Englewood Cliffs, N.J., 1992.
- , and Ware Myers: *Industrial Strength Software—Effective Management Using Measurement*, IEEE Press, Los Alamitos, Calif., 1997.
- Reifer, Donald (ed.): *Software Management*, 4th ed., IEEE Press, Los Alamitos, Calif., 1993.

- Rethinking the Software Process*, CD-ROM, Miller Freeman, Lawrence, Kans., 1996. (This CD-ROM is a book collection jointly produced by the book publisher, Prentice-Hall, and the journal publisher, Miller Freeman. It 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.)
- Rubin, Howard: *Software Benchmark Studies for 1997*, Howard Rubin Associates, Pound Ridge, N.Y., 1997.
- Roetzheim, William H., and Reyna A. Beasley: *Best Practices in Software Cost and Schedule Estimation*, Prentice-Hall PTR, Upper Saddle River, N.J., 1998.
- Stukes, Sherry, Jason Deshoretz, Henry Appgar, and Ilona Macias: *Air Force Cost Analysis Agency Software Estimating Model Analysis—Final Report*, TR-9545/008-2, Contract F04701-95-D-0003, Task 008, Management Consulting & Research, Inc., Thousand Oaks, Calif., September 1996.
- Stutzke, Richard D.: *Estimating Software-Intensive Systems: Projects, Products, and Processes*, Addison-Wesley, Boston, Mass, 2005.
- Symons, Charles R.: *Software Sizing and Estimating—Mk II FPA (Function Point Analysis)*, John Wiley & Sons, Chichester, U.K., 1991.
- Wellman, Frank: *Software Costing: An Objective Approach to Estimating and Controlling the Cost of Computer Software*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
- Yourdon, Ed: *Death March—The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*, Prentice-Hall PTR, Upper Saddle River, N.J., 1997.
- Zells, Lois: *Managing Software Projects—Selecting and Using PC-Based Project Management Systems*, QED Information Sciences, Wellesley, Mass., 1990.
- Zvegintzov, Nicholas: *Software Management Technology Reference Guide*, Dorset House Press, New York, 1994.