**Project planning and scheduling**

Software project management begins with a set of activities that are collectively called project planning. Before the project can begin, the manager and the software team must estimate the work to be done, the resources that will be required, and the time that will elapse from start to ﬁnish.

**5-1 Observation on Estimating**

Estimation of resources, cost, and schedule for a software engineering effort requires experience, access to good historical information, and the courage to commit to quantitative predictions when qualitative information is all that exists. Estimation carries inherent risk1 and this risk leads to uncertainty.

Project complexity has a strong effect on the uncertainty inherent in planning. Complexity, however, is a relative measure that is affected by familiarity with past effort.

Project size is another important factor that can affect the accuracy and efﬁcacy of estimates. As size increases, the interdependency among various elements of the software grows rapidly. Problem decomposition, an important approach to estimating, becomes more difﬁcult because decomposed elements may still be formidable.

The availability of historical information has a strong inﬂuence on estimation risk. By looking back, we can emulate things that worked and improve areas where problems arose. When comprehensive software metrics (Chapter 4) are available for past projects, estimates can be made with greater assurance, schedules can be established to avoid past difﬁculties, and overall risk is reduced.

**5-2 Project Planning Objectives**

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost, and schedule. These estimates are made within a limited time frame at the beginning of a software project and should be updated regularly as the project progresses.

The planning objective is achieved through a process of information discovery that leads to reasonable estimates. In the following sections, each of the activities associated with software project planning is discussed.

**5-3 Software Scope**

The ﬁrst activity in software project planning is the determination of software scope. Function and performance allocated to software during system engineering should be assessed to establish a project scope that is unambiguous and understandable at the management and technical levels. A statement of software scope must be bounded.

Software scope describes the data and control to be processed, function, performance, constraints, interfaces, and reliability. Functions described in the statement of scope are evaluated and in some cases reﬁned to provide more detail prior to the beginning of estimation. Because both cost and schedule estimates are functionally oriented, some degree of decomposition is often useful. Performance considerations encompass processing and response time requirements. Constraints identify limits placed on the software by external hardware, available memory, or other existing systems.

**5-4 Resources**

The second software planning task is estimation of the resources required to accomplish the software development effort. Figure 5.1 illustrates development resources as a pyramid. The development environment (hardware and software tools) sits at the foundation of the resources pyramid and provides the infrastructure to support the development effort. At a higher level, we encounter reusable software components (software building blocks that can dramatically reduce development costs and accelerate delivery). At the top of the pyramid is the primary resource—people.

Each resource is speciﬁed with four characteristics:

* Description of the resource.
* A statement of availability.
* Time when the resource will be required;
* Duration of time that resource will be applied.

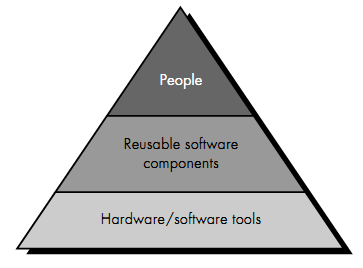


Figure 5.1 Project resources

**5-4-1 Human Resources**

The planner begins by evaluating scope and selecting the skills required to complete development. For relatively small projects (one person-year or less), a single individual may perform all software engineering tasks. The number of people required for a software project can be determined only after an estimate of development effort (e.g., person-months) is made. Techniques for estimating effort are discussed later in this chapter.

**5-4-2 Reusable Software Resources**

Component-based software engineering (CBSE) emphasizes reusability (the creation and reuse of software building blocks). Such building blocks, often called components, must be cataloged for easy reference, standardized for easy application, and validated for easy integration.

**5-4-3 Environmental Resources**

The environment that supports the software project, often called the software engineering environment (SEE), incorporates hardware and software. Hardware provides a platform that supports the tools (software) required to produce the work products that are an outcome of good software engineering practice.

A project planner must prescribe the time window required for hardware and software and verify that these resources will be available.

When a computer-based system (incorporating specialized hardware and software) is to be engineered, the software team may require access to hardware elements being developed by other engineering teams. For example, software for a numerical control (NC) used on a class of machine tools may require a speciﬁc machine tool (e.g., an NC lathe) as part of the validation test step; a software project for advanced page-layout may need a digital-typesetting system at some point during development. Each hardware element must be speciﬁed by the software project planner.

**5-5 Software Project Estimation**

In the early days of computing, software costs constituted a small percentage of the overall computer-based system cost. An order of magnitude error in estimates of software cost had relatively little impact. Today, software is the most expensive element of virtually all computer-based systems. For complex, custom systems, a large cost estimation error can make the difference between proﬁt and loss.

Software cost and effort estimation will never be an exact science. Too many variables—human, technical, environmental, political—can affect the ultimate cost of software and effort applied to develop it.

To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project (obviously, we can achieve

100% accurate estimates after the project are complete!).

2. Base estimates on similar projects that have already been completed.

3. Use relatively simple decomposition techniques to generate project cost and effort estimates.

4. Use one or more empirical models for software cost and effort estimation.

Decomposition techniques take a "divide and conquer" approach to software project estimation. By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.

Empirical estimation models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right. A model is based on experience (historical data) and takes the form



Where d is one of a number of estimated values (e.g., effort, cost, project duration) and vi are selected independent parameters (e.g., estimated LOC or FP).

**5-6 Decomposition Techniques**

Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece. For this reason, we decompose the problem, re-characterizing it as a set of smaller (and hopefully, more manageable) problems.

Lines of code and function points were described as measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function.

Baseline productivity metrics (e.g., LOC/pm or FP/pm) are then applied to the appropriate estimation variable, and cost or effort for the function is derived. Function estimates are combined to produce an overall estimate for the entire project.

Regardless of the estimation variable that is used, the project planner begins by estimating a range of values for each function or information domain value. Using historical data or (when all else fails) intuition, the planner estimates an optimistic, most likely, and pessimistic size value for each function or count for each information domain value. An implicit indication of the degree of uncertainty is provided when a range of values is speciﬁed. A three-point or expected value can then be computed. The expected value for the estimation variable (size), S, can be computed as a weighted average of the optimistic (sopt), most likely (sm), and pessimistic (spess) estimates. For example,

S = (Sopt + 4Sm + Spess)/6

**5-6-1 an Example of LOC-Based Estimation**

As an example of LOC and FP problem-based estimation techniques, let us consider a software package to be developed for a computer-aided design application for mechanical components. A review of the System Speciﬁcation indicates that the soft-ware is to execute on an engineering workstation and must interface with various computer graphics peripherals including a mouse, digitizer, high resolution color dis-play and laser printer.

Using the System Speciﬁcation as a guide, a preliminary statement of software scope can be developed:

*The CAD software will accept two- and three-dimensional geometric data from an engineer. The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good human/machine interface design. All geometric data and other supporting information will be maintained in a CAD database. Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter*

This statement of scope is preliminary—it is not bounded. Every sentence would have to be expanded to provide concrete detail and quantitative bounding. For example, before estimation can begin the planner must determine what "characteristics of good human/machine interface design" means or what the size and sophistication of the "CAD database" are to be.

For our purposes, we assume that further reﬁnement has occurred and that the following major software functions are identiﬁed:

• User interface and control facilities (UICF)

• Two-dimensional geometric analysis (2DGA)

• Three-dimensional geometric analysis (3DGA)

• Database management (DBM)

• Computer graphics display facilities (CGDF)

• Peripheral control function (PCF)

• Design analysis modules (DAM)

Following the decomposition technique for LOC, an estimation table, shown in Figure 5.3, is developed.

A range of LOC estimates is developed for each function. For example, the range of LOC estimates for the 3D geometric analysis function is:

* Optimistic—4600 LOC.
* Most likely—6900 LOC.
* Pessimistic—8600 LOC.

Applying Equation (5-1), the expected value for the 3D geometric analysis function is 6800 LOC. Other estimates are derived in a similar fashion.

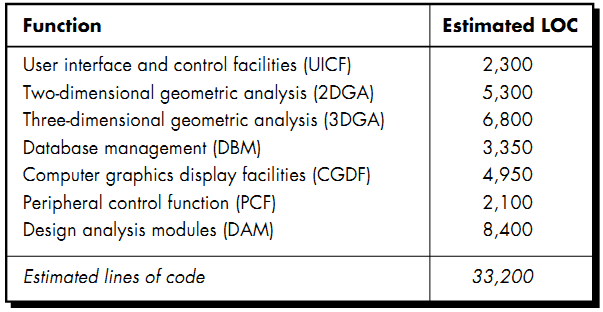


Figure 5-2 Estimation table for the LOC method

By summing vertically in the estimated LOC column, an estimate of 33,200 lines of code is established for the CAD system.

A review of historical data indicates that:

* the organizational average productivity for systems of this type is 620 LOC/pm.
* Based on a burdened labor rate of $8000 per month;
* the cost per line of code is approximately $13.

Based on the LOC estimate and the historical productivity data:

* the total estimated project cost is $431,000 .(13 $\* 33200)
* The estimated effort is 54 person-months.(33200/620)

5-6-2 an Example of FP-Based Estimation

Decomposition for FP-based estimation focuses on information domain values rather than software functions. Referring to the function point calculation table presented in Figure 5.4, the project planner estimates inputs, outputs, inquiries, ﬁles, and external interfaces for the CAD software. For the purposes of this estimate, the complexity weighting factor is assumed to be average. Figure 5.4 presents the results of this estimate.

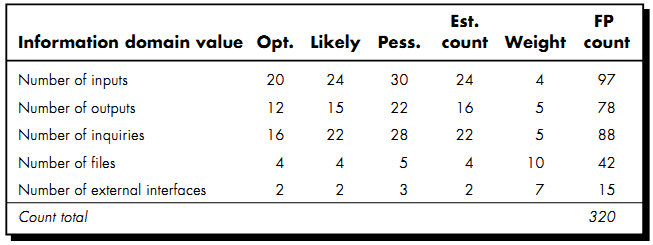
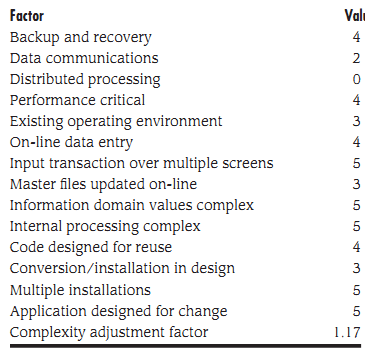


Figure 5-3 Estimating Information Domain Value

Each of the complexity weighting factors is estimated and the complexity adjustment factor is computed as described in Chapter 4:



Total 53.17

Finally, the estimated number of FP is derived:



The organizational average productivity for systems of this type is 6.5 FP/pm. Based on a burdened labor rate of $8000 per month, the cost per FP is approximately $1230.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is $461,000 and the estimated effort is 58 person-months.

**5-7 Empirical Estimation Models**

An estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP. Values for LOC or FP are estimated using the approach described in Sections 5.6.2 and 5.6.3. But instead of using the tables described in those sections, the resultant values for LOC or FP are plugged into the estimation model.

The empirical data that support most estimation models are derived from a limited sample of projects. For this reason, no estimation model is appropriate for all classes of software and in all development environments.

A typical estimation model is derived using regression analysis on data collected from past software projects. The overall structure of such models takes the form

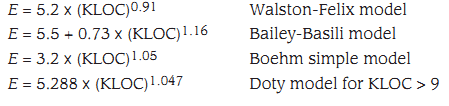
E = A + B X (ev)C …………………………….(5-2)

Where

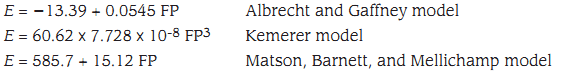
A, B, and C are empirically derived constants,

E is effort in person-months, and ev is the estimation variable (either LOC or FP).

In addition to the relationship noted in Equation (5-2), the majority of estimation models have some form of project adjustment component that enables E to be adjusted by other project characteristics (e.g., problem complexity, staff experience, development environment). Among the many LOC-oriented estimation models proposed in the literature are



FP-oriented models have also been proposed. These include



A quick examination of these models indicates that each will yield a different result14 for the same values of LOC or FP. The implication is clear. Estimation models must be calibrated for local needs!

**5-8 the Make/Buy Decision**

In many software application areas, it is often more cost effective to acquire than develop computer software. Software engineering managers are faced with a make/buy decision that can be further complicated by a number of acquisition options:

(1) Software may be purchased (or licensed).

(2) Software components (see may be acquired and then modified and integrated to meet specific needs.

(3) Software may be custom built by an outside contractor to meet the purchaser’s specifications.

In the ﬁnal analysis, the make/buy decision is made based on the following conditions:

(1) Will the delivery date of the software product be sooner than that for internally developed software?

(2) Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?

(3) Will the cost of out-side support (e.g., a maintenance contract) be less than the cost of internal support?

**5.8.1 Creating a Decision Tree**

The steps just described can be augmented using statistical techniques such as decision tree analysis. For example, Figure 5.6 depicts a decision tree for a software-based system, X. In this case, the software engineering organization can

(1) Build sys-tem X from scratch.

(2) Reuse existing “partial-experience” components to construct the system.

(3) Buy an available software product and modify it to meet local needs.

(4) Contract the software development to an outside vendor.

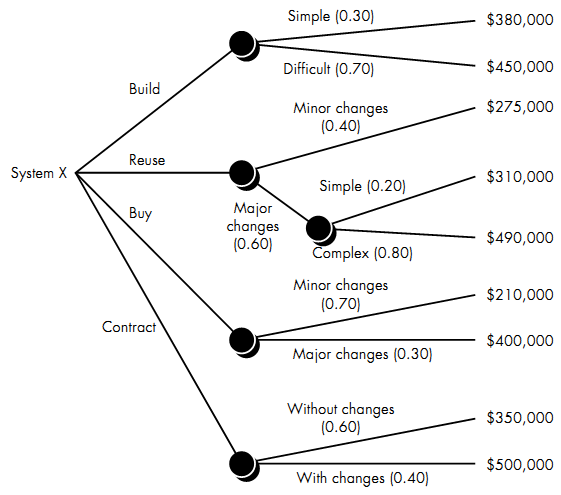


Figure 5-4 a decision tree to support the make/buy decision

If the system is to be built from scratch, there is a 70 percent probability that the job will be difﬁcult. Using the estimation techniques discussed earlier in this chapter, the project planner projects that a difﬁcult development effort will cost $450,000. A "simple" development effort is estimated to cost $380,000. The expected value for cost, computed along any branch of the decision tree, is



Where i is the decision tree path. For the build path

  
 Following other paths of the decision tree, the projected costs for reuse, purchase and contract, under a variety of circumstances, are also shown. The expected costs for these paths are

* Expected costreuse = 0.40 ($275K) + 0.60 [0.20($310K) + 0.80($490K)] = $382K
* Expected costbuy = 0.70($210K) + 0.30($400K)] = $267K
* Expected costcontract = 0.60($350K) + 0.40($500K)] = $410K

Based on the probability and projected costs that have been noted in Figure 5.6, the lowest expected cost is the "buy" option.