Chapter 2  Modeling the Process and Life Cycle

You can find out more information about LAI notation supporting tools from the International Software Process Constellation, Inc., in Reston, Virginia; you can send inquiries to 70160.142@computer.com.

The University of Southern California's Center for Software Engineering has developed a tool to assist you in selecting a process model suitable for your project's requirements and constraints. It can be found at http://usc.edu/pb/wb/ceseengineering/demonstration/laip.html, and more information can be found on the Center's Web site: http://usc.edu/cese.

Journals such as Software Process Improvement and Practice have articles addressing the role of process modeling in software development and maintenance. They also report the highlights of relevant conferences such as the International Software Process Workshop and the International Conference on Software Engineering. The July-August 2000 issue of IEEE Software offers an overview of diversity and has several articles about the success of a process maturity approach to software development.

2.12 EXERCISES

1. How does the description of a system relate to the notion of process models? For example, how do you decide what the boundary should be for the system described by a process model?

2. For each of the process models described in this chapter, what are the benefits and drawbacks of using the model?

3. For each of the process models described in this chapter, state what the model handles a significant change in an environment late in development?

4. Draw a diagram to capture the process of buying an airplane ticket for a business trip.

5. Draw a UML activity chart to define a model. Make sure that you include all the stages that show the model when it is untrusted, partially trusted, and completely trusted.

6. Use the notation of your choice, draw a process diagram of a software development process that prototypes three different designs and draws the tests from among them.

7. Explain the characteristics of good process models described in Section 2.4. Which characteristics are essential for processes to be used on projects where the problem and solution are not well understood?

8. In this chapter, we suggested that software development is a creation process, not a maintenance process. Discuss the characteristics of manufacturing processes that apply to software development and explain why characteristics of software development are more suitable for creative endeavors.

9. Should a development organization adopt a single process model for all of its software development? Discuss the pros and cons.

10. Suppose you are not comfortable with a particular software development process. How can you improve your understanding of the process?

11. Consider the models introduced in this chapter. Which one do you give the most flexibility to change in iteration-to-change-requirements?

12. Suppose Amalgamated, Inc., requires you to use a given process model when you work with a new system. You already building a system using the prescribed set of resources, and constraints. What are your suggestions for improving the process and what is the impact?
Chapter 7 Planning and Managing the Project

Answering the last two questions requires a well-thought-out project schedule. A project schedule describes the software development cycle for a particular project by calibrating the phases or stages of a project and breaking each into discrete tasks or activities to be done. The schedule also portrays the interactions among these activities and estimates the time that each task or activity will take. Thus, the schedule is a timeline that shows when activities will begin and end and when the release development process will be ready.

In Chapter 1, we learned that a systems approach involves both analysis and synthesis, breaking the problem into its component parts, designing a solution for each part, and then putting the pieces together to form a coherent whole. We can use this approach to determine the project schedule. We begin by working with customers and potential users to understand what they want and need. At the same time, we make sure that they are comfortable with our knowledge of their needs. We list all project deliverables that are the items that the customer expects to see during project development. Among the deliverables may be:

- documents
- demonstrations of function
- demonstrations of subsystems
- demonstrations of accuracy
- demonstrations of reliability, security, or performance

Next, we determine what activities must take place in order to produce these deliverables. We may use some of the process modeling techniques we learned in Chapter 2, laying out exactly what must happen and which activities depend on other activities, products, or requirements. Certain events are designated to be milestones, indicating our and our customers' expectations that a measurable level of progress has been made. For example, when the requirements are documented, inspected for consistency and completeness, and turned over to the design team, the requirements specification may be a project milestone. Similarly, milestones may include the completion of the user's manual, the performance of a given set of calculations, or a demonstration of the system's ability to communicate with another system.

In our analysis of the project, we must distinguish clearly between milestones and activities. An activity is a part of the project that takes place over a period of time, whereas a milestone is the completion of an activity—a particular point in time. That is, an activity has a beginning and an end, whereas a milestone is the end of a special designated activity. For example, the customer can see the system as being accomplishing an on-line operator's view. The development of the tutorial and its associated graphics is an activity. It demonstrates the tie between these functions to the customer at the milestone.

By examining the project carefully in this way, we can separate development into a succession of phases. Each phase is composed of steps and each step can be subdivided further if necessary as shown in Figure 3.1.

To see how this analysis works, consider the phases, steps, and activities of Table 3.1, which describes the building of a house. First, we consider two phases: landscaping us clearing and grubbing, seeding the yard, and planting trees and shrubs. Where necessary, we divide each phase into activities, for example, finishing the interior involves floor covering, doors, and fitments. Each activity is a measurable event and we have objective criteria to determine when the activity is complete. Thus, any activity's end can be a milestone, and Table 3.2 lists the milestones for phase 2.

This analytical breakdown gives us and our customers an idea of what is involved in constructing a house. Similarly, analyzing a software development is maintenance as a set of tasks for what is involved in developing and maintaining a system. We saw in Chapter 2 that a process model is a high-level view of the phases and steps, activities, both we and our customers have in mind. A process model in a high-level view of the phases and steps, activities, both we and our customers have in mind. A process model of a project is a high-level view of the phases and steps, activities, both we and our customers have in mind.
We can describe each activity with four parameters: the precedence, duration, date, and endpoint. A **predecessor** is an event or set of events that must occur before the activity can begin; it describes the set of conditions that allows the activity to begin. The **duration** is the length of time needed to complete the activity. The **date** is the date by which the activity must be completed, frequently determined by contractual deadlines. By specifying that the activity has ended, the **endpoint** is usually a milestone or deliverable.

We can illustrate the relationships among activities by using these parameters. In particular, we can draw an **activity graph** to depict the dependencies; the nodes of the graph are the project milestones, and the lines linking the nodes represent the activities involved. Figure 3.2 is an activity graph for the work described in phase 2 of Table 3.1.

Many important characteristics of the project are made visible by the activity graph. For example, it is clear from Figure 3.2 that neither of the two plumbing activities can start before milestone 2.2 is reached; that is, 2.2 is a predecessor to both interior and exterior plumbing. Furthermore, the figure shows us that several things can be done simultaneously. For instance, some of the interior and exterior activities are independent (such as installing wallboard, connecting external electrical wiring, and others leading to milestones 2.6 and 3.3, respectively). The activities on the left-hand path do not depend on those on the right for their initiation, so they can be worked on concurrently.

Notice that there is a dashed line from requiring permits (node 2.2) to surveying (node 1.1). This line indicates that those activities must be completed before excavation (the activity leading to milestone 1.3) can begin. However, since there is no activity that occurs after reaching milestone 1.2 in order to get to milestone 1.1, the line indicates a relationship without an accompanying activity.

It is important to realize that activity graphs depend on an understanding of the nature of tasks. If work cannot be done in parallel, then the (mostly straight) graph is misleading; that is, not reflecting the real-world constraints. Moreover, the graphs must depict a realistic depiction of the parallelism. In our house-building example, it is clear...
that some of the tasks, like plumbing, will be done by different people from those doing other tasks, like electrical work. But on software development projects, where some people have many skills, the theoretical parallelism may not reflect reality. A restricted number of people assigned to the project may result in the same person doing many things in series, even though they could be done in parallel by a larger development team.

**Estimating Completion**

We can make an activity graph more useful by adding to it information about the estimated time it will take to complete each activity. For a given activity, we label the corresponding edge of the graph with the estimate. For example, for the activities in phase 2 of the project, Table 3.1 shows the estimates for each activity.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Time Estimate (in Days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity 2.1: Install exterior plumbing</td>
<td>6</td>
</tr>
<tr>
<td>Activity 2.2: Install exterior electrical</td>
<td>8</td>
</tr>
<tr>
<td>Activity 2.3: Exterior siding</td>
<td>10</td>
</tr>
<tr>
<td>Activity 2.4: Install doors and windows</td>
<td>12</td>
</tr>
<tr>
<td>Activity 2.5: Install floor covering</td>
<td>15</td>
</tr>
<tr>
<td>Activity 2.6: Install doors and frames</td>
<td>18</td>
</tr>
</tbody>
</table>

Analyzing the path among the milestones of a project in this way is called the Critical Path Method (CPM). The path can show us the minimum amount of time it will take to complete the project, given our estimates of each activity's duration. Moreover, CPM reveals the activities that are most critical to completing the project on time.

To see how CPM works, consider again our house building example. First, we notice that the activity leading to milestones 2.1 (surveying) and 2.2 (requesting permits) occurs concurrently. Once the house is built, (the activity that culminates in milestones 2.1, or 2.3), the minimal construction begins. We cannot begin until both milestones 2.1 and 2.2 are reached. Excavation cannot begin until after the beginning of day 16. To make activity 3.1, or 3.2, the activity completion, must be completed before day 16. The house takes 15 days to complete, even though it only 3 days in duration. Thus, surveying has 15 days of available time, but requires only 3 days of real time. In the same way, for each activity in our graph, we can compute a pair of some real time and available time. The real time or actual time for an activity is the estimated amount of time required for the activity to be completed, and the available time is the amount of time available in the schedule for the activity's completion.
Section 3.1 Tracking Progress

Slack time or float for an activity is the difference between the available time and the real time for that activity:

Slack time = available time - real time

Another way of looking at slack time is to compare the earliest time an activity may begin with the latest time the activity may begin without delaying the project. For example, surveying may begin on day 1, to the earliest start time is day 1. However, because it will take 15 days to request and receive permits, surveying can begin on day 13 and still not delay the project schedule. Therefore:

Slack time = latest start time - earliest start time

Let us compute the slack for our example’s activities to see what it tells us about the project schedule. We compute slack by examining all paths from the start to the finish. As we have seen, it must take 15 days to complete milestones 1.1 and 1.2. An additional 35 days are used in completing milestones 1.3, 1.4, 2.1, and 2.2. At this point, there are four possible paths to be taken:

1. Following milestones 2.3 through 2.7 on the graph requires 39 days.
2. Following milestones 2.3 through 2.8 on the graph requires 42 days.
3. Following milestones 3.1 through 3.4 on the graph requires 54 days.
4. Following milestones 3.1 through 3.6 on the graph requires 54 days.

Because milestones 2.7, 2.8, 3.4, and 3.6 must be met before the project is finished, our schedule is constrained by the longest path. As you can see from Figure 3.3 and the preceding calculations, the two paths on the right require 124 days to complete, and the two paths on the left require fewer days. To calculate the slack, we can work backward along the path to see how much slack time is there for each activity leading to a node. First, we note that there is zero slack on the longest path. Then, we examine each of the remaining nodes to calculate the slack for the activities leading to them. For example, 34 days are available to complete the activities leading to milestones 2.3, 2.4, 2.5, 2.6, and 2.8, but only 42 days are needed to complete them. Thus, this portion of the graph has 12 days of slack. Similarly, the portion of the graph for activities 2.3 through 2.7 requires only 39 days, so we have 15 days of slack along this route. By working forward through the graph in this way, we can compute the earliest start time and slack for each of the activities. Then, we compute the latest start time for each activity by moving from the finish back through each node to this point. Table 3.4 shows the results: the slack time for each activity in Figure 3.3.

(A milestone 2.6, the path may be to milestones 2.7 or 2.8. The latest start times in Table 3.4 are calculated by using the route from 2.6 to 2.8, rather than from 2.6 to 2.7.)

The longest path has a slack of 0 for each of its nodes, because it is the path that determines whether or not the project is on schedule. For this reason, it is called the critical path.

Table 3.4 Slack Time for Project Activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Earliest Start Time</th>
<th>Latest Start Time</th>
<th>Slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>0</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>1.2</td>
<td>1</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>1.3</td>
<td>16</td>
<td>26</td>
<td>10</td>
</tr>
<tr>
<td>1.4</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>2.1</td>
<td>36</td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>2.2</td>
<td>51</td>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td>2.3</td>
<td>71</td>
<td>97</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>81</td>
<td>93</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>91</td>
<td>103</td>
<td>12</td>
</tr>
<tr>
<td>2.6</td>
<td>101</td>
<td>113</td>
<td>12</td>
</tr>
<tr>
<td>2.7</td>
<td>104</td>
<td>116</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>71</td>
<td>71</td>
<td>0</td>
</tr>
<tr>
<td>3.2</td>
<td>90</td>
<td>93</td>
<td>0</td>
</tr>
<tr>
<td>3.3</td>
<td>90</td>
<td>93</td>
<td>0</td>
</tr>
<tr>
<td>3.4</td>
<td>101</td>
<td>107</td>
<td>0</td>
</tr>
<tr>
<td>3.5</td>
<td>107</td>
<td>107</td>
<td>0</td>
</tr>
<tr>
<td>3.6</td>
<td>118</td>
<td>118</td>
<td>0</td>
</tr>
<tr>
<td>Finish</td>
<td>124</td>
<td>124</td>
<td>0</td>
</tr>
</tbody>
</table>
pat. Thus, the critical path is the one for which the slack at every node is zero. As you can see from our example, there may be more than one critical path. Since the critical path has no slack, there is no margin for error when performing the activities along its route.

Notice what happens when an activity on the critical path begins late (i.e., later than its earliest start time). The late start pushes all subsequent critical path activities forward, forcing them to be late, too, if there is no slack. And for activities not on the critical path, the subsequent activities may also lose slack time. Thus, the activity graph helps us understand the impact of any schedule slippage.

Consider what happens if the activity graph has several loops in it. Loops may occur when an activity must be repeated. For instance, in our house building example, the building inspector may require the plumbing to be re-inspected. In software development, the building inspector is required to re-inspect the plumbing. In software development, the building inspector is required to re-inspect the plumbing that has been re-inspected by the building inspector. In software development, the building inspector is required to re-inspect the plumbing that has been re-inspected by the building inspector.

Figure 3.4 is a bar chart that shows some software development project activities, including information about the duration of each activity, the earliest start date, and the latest start date. This chart is typical of charts produced by automated project management tools. The horizontal bars represent the duration of each activity, and the shaded areas represent the critical path. Activities depicted by dashed lines are on the critical path, and the duration is the earliest start time. Critical path analysis provides a project schedule to us who may wait for what we must do. The schedule is developed to complete all activities on schedule to ensure that the project is completed on time. This kind of analysis can be enhanced in many ways. For instance, our house building example suggests that we need to know exactly how long each activity will take. Often, this is not the case. Instead, we have only an estimated duration for each activity, based on our knowledge of similar projects and events. Thus, to each activity, we can assign a probable duration using some probability distribution, such that each activity has associated with it an expected value and a variance. In other words, instead of knowing an exact duration, we estimate a window of intervals within which the actual time is likely to fall. The expected value is a point within the interval, and the variance describes the width of the interval. You may be familiar with a standard probability distribution called the normal distribution, whose graph is a bell-shaped curve. The Program Evaluation Review Technique (PERT) is a popular critical path analysis technique that assumes a normal distribution. (See Hillier and Lieberman [1970] for more information about PERT.) PERT determines the probability that the earliest start time for an activity is close to the scheduled time.

Tools to Track Progress

There are many tools that can be used to keep track of a project's progress. Some are manual, others are software applications, and still others are sophisticated tools with complex graphs. To see what kinds of tools may be useful on your projects, consider the work breakdown structure depicted in Figure 3.5. Here, the overall objective is to build a system involving communications software, and the project manager has described the work in terms of five steps: system planning, system design, coding, testing, and delivery. For simplicity, we concentrate on the first two steps. Step 1 is then partitioned into five activities: reviewing the specifications, reviewing the budget, reviewing the schedule, reviewing the project plan, and reviewing the detailed design.

Figure 3.5 Work breakdown structure:

- System planning (1.0)
- System design (2.0)
  - Requirements specification (1.1)
  - Top-level design (2.1)
  - Detailed design (2.2)
- Coding (3.0)
- Testing (4.0)
- Delivery (5.0)
Section 3.1 Tracking Progress

Figure 3.7 Resource histogram.

Figure 3.6 Gantt chart for example work breakdown structure.
1.2 PROJECT PERSONNEL

To determine the project schedule and estimate the associated effort and costs, we need to know approximately how many people will be working on the project, what tasks they will perform, and what abilities and experience they must have so that they can do their jobs effectively. In this section, we look at how to decide who does what and how the staff can be organized.

Staff Holes and Characteristics

In Chapter 2, we examined several software process models, each depicting the way in which the several activities of software development are related. No matter the model, there are always activities necessary to any software project. For example, every project requires people to interact with the customers to determine what customers want, and by whom they want it. Other project personnel design the system, and still others write or test the programs. Key project activities are likely to include

1. requirements analysis
2. system design
3. program design
4. program implementation
5. testing
6. training
7. maintenance
8. quality assurance

However, not every task is performed by the same person or group; the assignment of staff to tasks depends on project size, staff expertise, and staff experience. There is great advantage in assigning different responsibilities to different sets of people, offering "checks and balances" that can identify faults early in the development process. For example, suppose the test team is separate from those who design and code the system. Testing new or modified software involves a system test, where the developers demonstrate to the customer that the system works as specified. The test team must define and document the criteria for linking the demonstrated functionality and performance characteristics in the requirements specified by the customer. The test team can then test the system according to the requirements documented, understanding how the internal pieces of the software are put together. Because the test team has no preconceptions about how the software works, it can concentrate on system functionality. This approach makes it easier for the test team to catch errors and omissions made by the developers or programmers. It is in part for this reason that the classroom method of using an independent test team, as we will see in later chapters (Mellinger, 1987).

For similar reasons, it is useful for program designers to be different from system designers. Program designers become deeply involved with the details of the code, and this sometimes negates the larger picture of how the system should work. We will discuss this in later chapters that techniques such as walkthroughs, inspections, and reviews bring the two types of designers together.

Section 3.2 Project Personnel

We saw in Chapter 1 that there are many other roles for personnel on the development or maintenance team. As we study each of the major tasks of development in subsequent chapters, we will describe the project team members who perform those tasks.

Once we have decided on the roles of project team members, we must decide which kinds of people we need in each role. Project personnel may differ in many ways, and it is not enough to say that a project needs an analyst, two designers, and five programmers, for example. Two people with the same job title may differ in at least one of the following ways:

- ability to perform the work
- interest in the work
- experience with similar applications
- experience with similar tools or languages
- experience with similar techniques
- experience with similar development environment
- training
- ability to communicate with others
- ability to share responsibilities with others
- management skills

Each of these characteristics can affect an individual's ability to perform productively. These variations help explain why one programmer can write a particular function in a day, whereas another requires a week. The differences can be critical, not only to schedule estimation, but also to the success of the project.

To understand each worker's performance, we must know his or her ability to perform the work at hand. Some are good at viewing the "big picture," but not interested in focusing on detail if asked to work on a small part of a large project. Such people may be better suited to system design or testing than to program design or coding. Sometimes, ability is related to context. In classes or on projects, you may have worked with people who are more comfortable programming in one language than another. Indeed, some developers feel more confident about their design abilities than their coding prowess. This feeling of comfort is important in the way they approach their work. They may have confidence in their ability to perform.

Interest in the work can also determine someone's success in a project. Although very good at doing a particular job, an employee may be more interested in trying something new than in repeating something done many times before. Thus, the novelty of the work is sometimes a factor in generating interest in it. On the other hand, there are always people who prefer doing what they know and do best, rather than venturing into uncharted territory. It is important that whatever is chosen for a task be exciting about performing it, no matter what the reason.

Given equal ability and interest, two people may still differ in their ability to explain or train in experience or training they have had with similar applications, tools, or techniques. The person who has already been successful at using C to write communications controller is more likely to write another communications controller in C faster than that not necessarily
more clearly or efficiently than someone who has never been involved in the development of the system. This is why selection of project personnel involves not only individual ability and skill, but also experience and training.

On every software development or maintenance project, members of the development team communicate with one another, with users, and with the customer. The project's progress is affected not only by the degree of communication, but also by the ability of individuals to communicate their ideas. Software failures can result from a breakdown in communication and understanding, so the number of people who need to be able to communicate with one another can affect the quality of the resulting product. If a project has $n$ workers, then there are $\binom{n}{2}$ pairs of people who might need to communicate. Each project involves only 10 people can use 45 lines of communication, and there are 1023 possible committees or teams that can be formed to handle subsystem development.

Many projects involve several people who must share responsibility for completing one or more activities. Those working on one aspect of project development must trust the other team members to do their parts. In classes, you are usually in total control of the project, but on a software project, you must trust your colleagues to do their work. If you are working on a team, you must be able to share the workload. Not only does this require cooperation and trust, but also trust in your colleagues to do the work you have assigned.

Communication is an issue in managing the project. Some people are good at directing the work of others. This aspect of personal interaction is also related to the comfort people feel in their jobs. Those who feel uncomfortable with the idea of pushing their colleagues to stay on schedule, to document their code, or to meet with the customer are not good candidates for development jobs involving the management of others.

**Figure 3.5** Communication paths on a project

**Sidebar 3.1 Make Meetings Enhance Project Progress**

Scheduling time for meetings is a challenge for project managers. One of the challenges is to ensure that the meetings are productive. If the meetings are not productive, it can be very difficult to achieve the goals of the project. Therefore, it is important to schedule the meetings wisely. The meetings should be scheduled to ensure that they are productive.

**Work Styles**

Different people have different preferred styles for interacting with others on the job. You may want to consider the following questions when deciding on your preferred work style:

- Do you prefer to: do detailed analytical work, or do you prefer to work with others?
- Do you prefer to: work alone, or do you prefer to work in a group?
- Do you prefer to: work at a steady pace, or do you prefer to work quickly?
- Do you prefer to: work in a structured environment, or do you prefer to work in a flexible environment?
- Do you prefer to: work in a team, or do you prefer to work individually?
not need to see documents or hear explanations supporting with option. If her time is wasted or her efficiency is hampered in some way, she asserts her authority to regain control of the situation. Thus, Kai is good at making sound decisions quickly.

Marcel, a rational introvert, is very different from his colleague Kai. He judges his peers by how busy they are, and he has little tolerance for those who appear not to be working hard all the time. He is a good worker, admired for the energy he devotes to his work. His reputation as a good worker is very important to him, and he presents himself on being accurate and thorough. He does not make decisions without complete information. When asked to make a presentation, Marcel does so only after gathering all relevant information on the subject.

Marcel shares an office with an intuitive introvert, Whereas Marcel will not make a decision without complete knowledge of the situation. Marcel prefers to follow his feelings. Often, he will trust his intuition about a problem, basing his decision on professional judgment rather than a slow, careful analysis of the information at hand. Since he is intuitive, David tends to tell the others on his project about his new ideas. He is creative, and he enjoys when others recognize his ideas. David likes to work in an environment where there is a great deal of interaction among the staff members.

Ying, an intuitive introvert, also thrives on her colleagues’ attention. She is sensitive and aware of her emotional reactions to people and problems. It is very important that she is liked by her peers. Because she is a good listener, Ying is the project member to whom others turn to express their feelings. Ying takes a lot of time to make a decision, not only because she needs complete information, but also because she wants to make the right decision. She is sensitive to what others think about her ability and ideas. She analyzes situations much as Marcel does, but with a different focus. Marcel looks at all the facts and figures, but Ying examines emotional dependencies and emotional involvements too.

Clearly, not everyone fits neatly into one of the four categories. Different people have different tendencies, and we can use the framework of Figure A10 to describe these tendencies and preferences.

Communication is critical to project success, and work style determines communication style. For example, if you are responsible for a part of the project that is behind schedule, Kai and David are likely to tell you when your work must be ready. David may offer several ideas to get the work back on track, and Kai will give you a new schedule to follow. However, Marcel and Ying will probably ask when the results will be ready. Marcel, in analyzing his options, will want to know why it is not ready; Ying will ask if there is anything she can do to help.

Understanding work styles can help you to be flexible in your approach to other project team members and to subordinates and users. In particular, work styles give you information about the priorities of others. If a colleague’s priorities and interests are different from yours, you can present information to her in terms of what she deems important. For example, if you are preparing a presentation for a customer, you know that the customer is interested in hearing things that are important to that person. If you are preparing a presentation for a customer, you know that she will likely consider the arguments that are important to her. Thus, you can organize your presentation so that it tells her what is important, and how the project is progressing. However, if the customer is an employee, you can include questions on whether the job was done well or not. Similarly, if the customer is an employee, you can take advantage of his creativity by soliciting new ideas from him; if he is rational, your presentation can...
structure minimizes the amount of communication needed during the project. Each team member must communicate often with the chief, but not necessarily with other team members. Thus, if the team consists of \( n \) programmers plus the chief, the team can establish only \( n + 1 \) paths of communication (one path for each team member's interaction with the chief) out of a potential \( n(n + 1)/2 \) paths. For example, rather than working out a problem themselves, the programmers can simply approach the chief for an answer. Similarly, the chief reviews all design and code, removing the need for peer reviews.

Although a chief programmer team is a hierarchy, groups of workers may be formed to accomplish a specialized task. For instance, one or more team members may form an administrative group to provide a status report on the project's current cost and schedule.

Clearly, the chief programmer must be good at making decisions quickly, so the chief is likely to be an extrovert. However, if most of the team members are introverts or non-introverts, the chief programmer team may not be the best structure for the project. An alternative is based on the idea of "egoless" programming, described by Weinberg (1971). Instead of a single point of responsibility, an egoless approach holds everyone equally responsible. Moreover, the process is separated from the individual contributions made to the project or the results, not the people involved. The egoless team structure is democratic and all team members vote on a decision, whether they concern design considerations or testing techniques.

Of course, there are many other ways to organize a development or maintenance project, and the two described here represent extremes. Which structure is preferable? The more people on the project, the more need there is for a formal structure. Certainly, a development team with only three or four members does not always need an elaborate organizational structure. However, a team of several dozen workers must have a well-defined organization. In fact, your company or your customer may impose a structure on the development team, based on past success, on the need to track progress in a certain way, or on the desire to minimize points of contact. For example, your customer may insist that the test team be totally independent of program design and development.

Researchers continue to investigate how project team structure affects the result of planning and how to choose the most appropriate organization in a given situation. A National Science Foundation (1983) investigation found that projects with a high degree of certainty, stability, uniformity, and repetition can be accomplished more effectively by a hierarchical organizational structure such as the chief programmer team. These projects require little communication among project members, so they are well-suited to an organization that stresses rules, specialization, formality, and a clear definition of organizational hierarchy.

On the other hand, when there is much uncertainty involved in a project, a more democratic approach may be better. For example, if the requirements may change as development proceeds, the project has a degree of uncertainty. Likewise, if the customer is building a new piece of hardware to interface with a system, if the exact specification of the hardware is not yet known, then the level of uncertainty is high. Here, participation in decision making, a loosely defined hierarchy, and the encouragement of open communication can be effective.
Table 3.5 summarizes the characteristics of projects and the suggested organizational structure to address them. A large project with high certainty and repetition problems needs a highly structured organization, whereas a small project with few techniques requires a loosely structured and highly certain structure. Sidebar 3.2 describes the need for a team to balance structure with creativity.

The two types of organizational structure can be combined, where appropriate. For instance, programmers may be asked to develop a subsystem on their own, using an egoless approach within a hierarchical structure. Or the test team of a loosely structured group within a hierarchical structure can be responsible for all major testing decisions.

3.3 EFFORT ESTIMATION

One of the critical aspects of project planning and management is understanding how much time is required to complete a project. Cost overruns can cause customers to cancel projects, and cost underestimates can force a project team to invest much of its time without financial compensation. As described in Sidebar 3.3, there are many reasons for inaccurate estimates. A good cost estimate early in the project's life helps the project manager to know how many developers will be required, and to arrange for the appropriate staff to be available when they are needed.

SIDEBAR 3.3 CAUSES OF INACCURATE ESTIMATES

Liker and Prasad (1992) interviewed cost-estimation practices of 115 different organizations. Thirty-five percent of the managers surveyed on a five-point Likert scale indicated that their current estimates were "moderately unsatisfactory" or "very unsatisfactory." The key causes identified by the respondents included:

- Frequent requests for changes by users
- Overlooked tasks
- Users' lack of understanding of their own requirements
- Insufficient analysis when developing an estimate
- Lack of coordination of systems development, technical services, operations, data administration, and other functions during development
- Lack of an adequate method or guidelines for estimating

Several aspects of the project were assessed to determine whether they influenced the estimate:

- Complexity of the proposed application system
- Required integration with existing systems
- Size of the system expressed as number of functions or programs
- Capabilities of the project team members
- Project team experience with the application
- Anticipated frequency or extent of potential changes in user requirements
- Project team's experience with the programming language
- Database management system
- Number of project team members
- Extent of programming or documentation standards
- Availability of tools such as application generators
- Team's experience with the hardware
The project budget pays for several types of costs: facilities, staff, materials, and tools. The facility costs include hardware, space, furniture, telephone, motors, heating and air conditioning, cable, disks, paper, pens, telephones, and all other items that provide the physical environment in which the developers work. For some projects, this environment may already exist; on the other hand, easy to estimate. But for other projects, the environment must be created. For example, a new project may require a conference room, library, or special equipment. In this case the costs can be estimated, but the time to estimate costs may vary from initial estimates to final estimates. Consequently, inserting columns in a building may seem straightforward until the builders discover that the building is of special historical significance, so that the costs must be divided around the walls instead of through them.

These are sometimes hidden costs that are not apparent to the managers and developers. For example, a study indicated that a programmer needs a minimum amount of space and quiet to be able to work effectively. M. J. C. (1978) reported to his college at IBM that the minimum standard for programmer work space should be 100 square feet of dedicated floor space with 10 square feet of horizontal work surface. The space also needs a floor level that is comfortable for those who are sitting at it. Sitting on telephone calls and uninvited visitors are more efficient and produce a better product than those who are subject to repeated interruption.

Other project costs are hardware, software, and tools to support development efforts. In addition to tools for designing and coding the system, the project may buy software to capture requirements, design, documentation, test the code, keep track of changes, generate test data, support group meetings, and more. These tools are sometimes calculated as Computer-Aided Software Engineering (CASE) tools and are sometimes required by the customer or in part of a company's usual software development process.

For most projects, the largest component of costs is labor. We must determine how much of this cost will be required to complete the project. First we need to estimate the total amount of the project component is the largest portion of the project execution cost. We have seen that as you spend more on labor, the project organization, payroll, interest, experience, training, and other factors can affect the time it takes to complete a task. Moreover when a group of workers must communicate and consult with one another, the cost is increased, especially if they need to be located in different places.

Cost, schedule, and cost estimation must be done as early as possible during the project life cycle, since it affects resource allocation and project feasibility. If costs are too high, the customers may cancel the project. Budget estimation should be done repeatedly throughout the project life cycle. As the project changes, the cost estimate can be refined, based on more complete information about the project's characteristics and size. It is therefore important to update the cost estimates frequently (Boehm et al. 1995).

The stars represent size estimates from actual projects, and the plus are cost estimates. The funnel-shaped lines surrounding the right represent Johnson's sense of how our estimates get more accurate as we learn more about a project. Notice that when the size of the project is very large, the estimate can differ from the eventual actual cost by a factor of 4. Some decisions are made about the product and the process, the
our estimate is the mean of the beta probability distribution determined by these numbers: \( 1 \pm 4 \times 2 \). By using this technique, we produce an estimate that "normalizes" the individual estimates.

The Delphi technique makes use of expert judgment in a different way. Experts are asked to make individual predictions secretly, based on their expertise and using whatever process they choose. Then, the average estimate is calculated and presented to the group. Each expert has the opportunity to revise his or her estimate, if desired. The group process is repeated until no expert wants to revise. Some users of the Delphi technique process is resubmitted when no experts want to revise. At other times, the users allow no discussion among the variations, judicious use of the expert is determined anonymously among the experts.

Wallerstein (1974) built one of the first models of software development effort. His software cost model captures his experience with project cost at TRW, a U.S. software development company. As shown in Table 3.6, the row name represents the type of software component. Difficulty depends on two factors: whether the column contains a 0, and, whether the column contains a 0, and, whether the column contains a 0, and, whether the column contains a 0, and, whether the column contains a 0, and, whether the column contains a 0. The matrix elements are the cost per line of code, as calculated from historical data at TRW. To use the matrix, you partition the proposed software system into modules. Then, you estimate the size of each module in terms of lines of code. Using the matrix, you can use the cost per module, and then sum all the modules. For instance, suppose you have a system with three modules: one input/output module that is old and easy, one algorithm module with that is old and hard, and one data management module that is old and moderate. If the model estimates the cost to be \( (100 \times 10) + (200 \times 35) + (100 \times 31) \), it is found to be approximately \$11,900.

Since the model is based on TRW data and uses 1974 dollars, it is not applicable to today's software development projects. But the technique is useful and can be adapted easily to your own development or maintenance environment.

In general, experimental models, by relying mostly on expert judgment, are subject to all its inaccuracies. They rely on the expert's ability to determine which projects are similar and in what ways. However, projects that appear to be very similar \( x \) in fact be quite different. For example, fast runners today can run a mile in 4 minutes. A marathon race requires a runner to run 26 miles and 365 yards. If we extrapolate the 4-minute time, we might expect a runner to run a marathon in 1 hour and 45 minutes. Yet a marathon has never been run in under 2 hours. Consequently, there must be characteristics of running a marathon that are very different from those of running a mile. Likewise, there are often characteristics of one project that make it very different from another project, but the characteristics are not always apparent.

Even when we know how one project differs from another, we do not always know how the differences affect the cost. A proportional strategy is unreliable, because project costs are not always linear. Two people cannot produce code twice as fast as one. Extra time may be needed for communication and coordination, to accommodate differences in interest, ability, and experience. Scattergood, Everson, and Grant (1968) found that the productivity ratio between best and worst programmers averaged 10 to 1, with no easily definable relationship between experience and performance. Likewise, a recent study by Hughes (1998) found great variability in the way software is designed and developed, so that a model that may work in one organization may not apply to another. Hughes also noted that past experience and knowledge of available resources are major factors in determining cost.

Expert judgment suffers not only from variability and subjectivity, but also from dependence on current data. The data on which an expert judgment model is based must reflect current practices, so it must be updated often. Moreover, most expert judgment techniques are simplistic, neglecting to incorporate a large number of factors that can affect the effort needed on a project. For this reason, practitioners and researchers have turned to algorithmic methods to estimate effort.

### Algorithmic Methods

Researchers have created models that express the relationship between effort and the factors that influence it. The models are usually described using equations, where effort is the dependent variable, and several factors (such as experience, size, and application type) are the independent variables. Most of these models acknowledge that the project size is the most influential factor in this equation by expressing effort as

\[
E = (a + bS)mX
\]

where \( S \) is the estimated size of the system, and \( a, b, \) and \( c \) are constants. \( X \) is a vector of cost factors, \( c^1 \) through \( c^m \), and \( m \) is an adjustment multiplier based on these factors. In other words, the effort is determined mostly by the size of the proposed system, adjusted by the effects of several other project, process, product, or resource characteristics.

Wallerstein and Felix (1977) developed one of the first models of this type, finding that IBM data from 62 projects yielded an equation of the form

\[
E = 5.25S^{1.5}
\]
The projects that supplied data built systems with size ranging from 5000 to 457,000 lines of code, written in 28 different high-level languages on 66 computers, and representing from 1.2 to 11.75% person-months of effort. Size was measured as lines of code, including comments as long as they did not exceed 50% of the total lines in the program.

The basic equation was supplemented with a productivity index that reflected 79 factors that can affect productivity, shown in Table 3.7. Notice that the factors are tied to a very specific type of development, including two platforms: an operational computer and a development computer. The model reflects the particular development style and the development phase of the IBM Federal System organization, that provided the data.

Each of the 29 factors was weighted by 1 if the factor increases productivity, 0 if it has no effect on productivity, and -1 if it decreases productivity. A weighted sum of the 29 factors was then used to generate an effort estimation from the basic equation.

### Table 3.7: Walter and Felix Model Productivity Factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Computer interface complexity</td>
</tr>
<tr>
<td>2.</td>
<td>User participation in requirements definition</td>
</tr>
<tr>
<td>3.</td>
<td>Custom-oriented program design change</td>
</tr>
<tr>
<td>4.</td>
<td>Consultant experience with the application area</td>
</tr>
<tr>
<td>5.</td>
<td>Overall personnel experience</td>
</tr>
<tr>
<td>6.</td>
<td>Percentage of development programmers who participated in the design of functional specifications</td>
</tr>
<tr>
<td>7.</td>
<td>Previous experience with the operational computer</td>
</tr>
<tr>
<td>8.</td>
<td>Previous experience with the programming language</td>
</tr>
<tr>
<td>9.</td>
<td>Previous experience with applications of similar size and complexity</td>
</tr>
<tr>
<td>10.</td>
<td>Ratio of average staff size to project duration (people per month)</td>
</tr>
<tr>
<td>11.</td>
<td>Number of computer operators under computer development</td>
</tr>
<tr>
<td>12.</td>
<td>Access to development computer open or special request</td>
</tr>
<tr>
<td>13.</td>
<td>A-cells to development computer closed</td>
</tr>
<tr>
<td>14.</td>
<td>Classified security environment for computer and at least 25% of programs and data</td>
</tr>
<tr>
<td>15.</td>
<td>Use of structured programming</td>
</tr>
</tbody>
</table>

Bailey and Basili (1981) suggested a modeling technique, called a meta-model, for building an estimating equation that reflects your own organization's characteristics. They demonstrated their technique using a database of 18 scientific projects written in Fortran at NASA's Goddard Space Flight Center. First, they minimized the standard error estimate and produced an equation that was very accurate:

\[ E = 5.5 + 0.735R \]

Then, they adjusted this initial estimate based on the ratio of errors, if \( R \) is the ratio between the actual effort, \( E \), and the predicted effort, \( E \), then the effort adjustment is defined as

\[ ER_a = \begin{cases} R - 1, & \text{if } R \geq 1 \\ 1 - 1/R, & \text{if } R < 1 \end{cases} \]

They then adjusted the initial effort estimate, \( E \), this way:

\[ E_{adj} = \frac{1}{1 + ER_a} \cdot E \]

Finally, Bailey and Basili (1981) accounted for other factors that affect effort, shown in Table 3.8. For each entry in the table, the project is scored from 0 (not present) to 5 (very important), depending on the judgment of the project manager. Thus, the total

### Table 3.8: Bailey-Basili Effort Model

<table>
<thead>
<tr>
<th>Model Methodology (METH)</th>
<th>Cumulative Complexity (CPLX)</th>
<th>Cumulative Experience (EXP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree chart</td>
<td>Customer interface complexity</td>
<td>Programmer qualifications</td>
</tr>
<tr>
<td>Top-down design</td>
<td>Application complexity</td>
<td>Programmer readiness experience</td>
</tr>
<tr>
<td>Formal documentation</td>
<td>Program flow complexity</td>
<td>Programmed language experience</td>
</tr>
<tr>
<td>Chief programmer team</td>
<td>Journalistic communication complexity</td>
<td>Programmer application experience</td>
</tr>
<tr>
<td>Formal training</td>
<td>Database complexity</td>
<td>Team experience</td>
</tr>
<tr>
<td>Formal test plan</td>
<td>External communication complexity</td>
<td></td>
</tr>
<tr>
<td>Design formulations</td>
<td>Customer-oriented program, design change</td>
<td></td>
</tr>
<tr>
<td>Code readable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code development folders</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
score for MESH can be as high as 45, for CPLEX as high as 35, and for EXP as high as 25. Their model describes a procedure, based on multivariate least-squares regression, for using these scores to further modify the effort estimate.

Clearly, one of the problems with models of this type is that they depend on size as a key variable. Estimates are usually required early, well before accurate size information is available, and certainly before the system is expressed as lines of code. So the models must incorporate the effort estimation problem as a size estimation problem.

Booch (1981) developed the original COCOMO model in the 1970s, using an extensive database of information from projects at IBM, an American company that builds extensive databases of information from projects at other companies. This information was collected as part of the project planning process, and the project planning process itself was developed to be done in a systematic way.

COCOMO II, Booch's updated model, was released in 1994. It builds on the original COCOMO model, but it includes several important improvements. The most important of these is that it includes a more detailed description of the software development process, and it includes more detailed information about the size of the software system. COCOMO II also includes more detailed information about the size of the software system, and it includes more detailed information about the size of the software system, and it includes more detailed information about the size of the software system.

The COCOMO II estimation process reflects three major stages of any development project. There are three major stages of any development project. The first stage is the planning stage, where the size of the project is estimated. The second stage is the design stage, where the system is designed. The third stage is the implementation stage, where the system is implemented.

The COCOMO II model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model. The model is based on a linear regression model.

### Table 3.3: New Stages of COCOMO II

<table>
<thead>
<tr>
<th>Model Aspect</th>
<th>Stage 1: Application Composition</th>
<th>Stage 2: Early Design</th>
<th>Stage 3: Postarchitecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Application points</td>
<td>Location points (P's)</td>
<td>FP and runtime of some lines of code (LOC)</td>
</tr>
<tr>
<td>Resource</td>
<td>Higher at n model</td>
<td>Equations of SLOC in function of other variables</td>
<td>Equations of LOC in function of other variables</td>
</tr>
<tr>
<td>Reusability</td>
<td>Implied prior model</td>
<td>% change of model</td>
<td>% change of model</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Application Points</td>
<td>Function of 21, software interfaces, understanding, availability</td>
<td>Function of 21, software interfaces, understanding, availability</td>
</tr>
</tbody>
</table>

### Scaling 1/2 or meaningful effort estimates

- 50: 0.9 to 1.3, depending on processor speed, software interface, early architecture, risk assessment, non-code, and non-process maturity

### Production cost drivers

- None: Computed, required feasibility
- Platform cost drivers: None: Platform difficulty
- Personnel cost drivers: None: Personnel capability and experience

### Project cost drivers

- None: Required development schedule, non-development

At stage 1, application points supply the size measure. This size measure is an extension of the object-oriented approach suggested by Kaufman and Kuester (1983) and productivity data reported by Banker, Kaufman, and Kuester (1984). To compute application points, you first count the number of screens, reports, and third generation language components that will be involved in the application. It is assumed that these
elements are defined in a standard way as part of an integrated computer-aided software engineering environment. Next, you classify each application element as simple, medium, or difficult. Table 3.10 contains guidelines for this classification.

The numbers in Table 3.10 are the same as the numbers in the previous section. The weight for each application element is calculated by using the following formula:

New application points = (application points) × (100 / 100)

To use this number for effort estimation, you use an adjustment factor, called a productivity rate, based on developers' experience and capability, coupled with CASE maturity and capability. The productivity rate is a function of the number of person-months required to develop an application element. The number of person-months required is the number of productivity factors in Table 3.12. Table 3.12 lists the number of person-months required for each level of productivity factor. For example, if the productivity factor is 7, then the number of person-months required is 7. The developers' experience and capability are also considered. The productivity rate is a function of the number of person-months required for the application element. The productivity rate is a function of the number of person-months required for the application element. The productivity rate is a function of the number of person-months required for the application element.

At stage 1, the effort drivers are not applied to the gross effort estimate. However, at stage 2, the effort estimate based on function points calculation is adjusted for dependency on reuse, requirements, and maintenance. The scale factor is the value for c in the effort equation. The scale factor is known as the productivity factor. The productivity factor is a function of the number of person-months required for the application element. The productivity factor is a function of the number of person-months required for the application element. The productivity factor is a function of the number of person-months required for the application element.

The cost drivers in Table 3.12 are adjusted factors, expressed as effort multipliers. The cost drivers in Table 3.12 are adjusted factors, expressed as effort multipliers. The cost drivers in Table 3.12 are adjusted factors, expressed as effort multipliers.

### Table 3.11: Complexity Weights for Application Points

<table>
<thead>
<tr>
<th>Element type</th>
<th>Simple</th>
<th>Medium</th>
<th>Difficult</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Report</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>SQL component</td>
<td></td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

### Table 3.12: Producibility Factor Estimation

<table>
<thead>
<tr>
<th>Driver</th>
<th>Very high</th>
<th>High</th>
<th>Nominal</th>
<th>Low</th>
<th>Very low</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE maturity</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Productivity</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Scale factor</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Note: The productivity factor is the value for c in the effort equation. The scale factor is known as the productivity factor. The productivity factor is a function of the number of person-months required for the application element.
Machine Learning Methods

In the past, most effort and cost modeling techniques relied on analytic methods. That is, researchers have examined data from past projects and generated equations. This is different from the machine learning techniques that are becoming more popular in the software engineering community. These techniques involve looking back at what has happened in the past and using this information to make predictions about future projects. The basic idea is to use a set of rules or equations to predict the effort required for a project.

Several researchers have used back-propagation algorithms on similar neural networks to predict development effort, including estimation for projects using fourth-generation languages (Wirtz and Finnie 1984; Srinivasan and Fisher 1995; Srinivasan, Ellison, and Dagdug 1997). Shepperd (1997) reports that the accuracy of this type of model seems to be sensitive to details about the topology of the neural network, the number of learning stages, and the initial weights of the neurons within the network. The networks also seem to require large training sets in order to give good predictions. On the other hand, they may be based on a great deal of experience, rather than a few representative projects. This type of model is sometimes difficult to obtain, especially collected consistently in large quantity, so the accuracy of these models is hard to estimate. However, if the model produces more accurate estimates, organizations may be more willing to collect data for the networks.

In general, this "learning" approach has been tried in different ways by other researchers. Srinivasan and Fisher (1995) used Kemerer's data (Kemerer 1989) with a statistical technique called regression trees; they produced predictions more accurate than those of the original COCOMO model and SLIM, a proprietary commercial model. However, their results were not as good as those produced by a neural network or a model based on function points. Brand, Bissell, and Thomas (1992) obtained better results from using a tree induction technique, using the Kemerer and COCOMO datasets. Porter and Selby (1990) also used a tree-based approach; they constructed a decision tree that identifies which project, process, and product characteristics are most important. The model is then used to predict effort for new projects.
formation about past projects. Shepperd (1997) points out that CBR offers two clear advantages over many of the other techniques. First, CBR deals only with events that actually occur, rather than with the much larger set of all possible occurrences. This same feature also allows CBR to deal with poorly understood domains. Second, it is easier for users to understand particular cases than to depict events as chains of rules or as ordered networks.

Estimation using CBR involves four steps:
1. The user identifies a new problem as a case.
2. The system retrieves similar cases from a repository of historical information.
3. The system reuses knowledge from previous cases.
4. The system suggests a solution for the new case.

The solution may be revised, depending on actual events, and the outcome is placed in the repository, building up the collection of completed cases. However, there are two big hurdles to creating a successful CBR system: characterizing cases and determining similarity.

Cases are characterized based on the information that happens to be available. Usually, experts are asked to supply a full set of features that are significant in describing a case, and in particular in determining when two cases are similar. In practice, similarity is usually measured using an n-dimensional vector of features. Shepperd, Schofield, and Kitchinham (1996) found a CBR approach to be more accurate than traditional regression analysis-based algorithmic methods.

Finding the Model for Your Situation

There are many effort and cost models being used today: commercial tools based on past experience or intricate models of development, and some are tools that access databases of historical information about past projects. Validating these models (i.e., making sure the models reflect actual practice) is difficult, because a large amount of data is needed. An important criterion is the validity of the developed model. Moreover, if a model fails to explain a large and varied set of situations, the supporting database must include measures from a very large and varied set of development environments.

Even when you find models that are designed for your development environment, you must be able to evaluate which are the most accurate on your project. There are two sets of models that are often used to help you in assessing the accuracy. MMRE and PMRE, PRED (PRED) is the percentage of projects for which the estimated effort is within 5% of the actual value. For most effort, cost, and schedule models, managing within 5% of the actual value is very difficult, and those models whose estimates are within 25% of the actual value are considered to perform well if PRED (PRED) is greater than 75%. A model is considered to function well if PRED (PRED) is greater than 75%.

Table 3.14 Summary of Model Performance

<table>
<thead>
<tr>
<th>Model</th>
<th>PRED (25)</th>
<th>MMRE</th>
</tr>
</thead>
<tbody>
<tr>
<td>White-Russi</td>
<td>0.30</td>
<td>0.45</td>
</tr>
<tr>
<td>Basic COCOMO</td>
<td>0.27</td>
<td>0.60</td>
</tr>
<tr>
<td>Intermediate COCOMO</td>
<td>0.85</td>
<td>0.22</td>
</tr>
<tr>
<td>Intermediate COCOMO (variation)</td>
<td>0.78</td>
<td>0.19</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.78</td>
<td>0.68</td>
</tr>
<tr>
<td>Pfeiffer</td>
<td>0.50</td>
<td>0.29</td>
</tr>
<tr>
<td>SIML</td>
<td>0.76-0.74</td>
<td>0.78-1.04</td>
</tr>
<tr>
<td>Jensen</td>
<td>0.76-0.85</td>
<td>0.70-1.18</td>
</tr>
<tr>
<td>COCOMO</td>
<td>0.30-0.43</td>
<td>0.23-0.57</td>
</tr>
<tr>
<td>General COCOMO</td>
<td>0.76</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Estimation for all types of development. However, the relationships among cost factors are not simple, and the models must be flexible enough to handle changing use of tools and methods.

Moreover, Kitchinham, MacDonald, and Shepperd (2000) point out that the MMRE and PRED statistics do not directly measure estimation accuracy. They suggest that you use the simple ratio of estimate to actual estimate/actual. This measure has a distribution that closely reflects estimation accuracy. By contrast, MMRE and PRED are measures of the spread (standard deviation) and precision (kurtosis) of the ratio, so they tell you only characteristics of the distribution.

Even when estimation models provide reasonably accurate estimates, we must be able to understand which types of effort are needed during development. For example, designers may not be needed until the requirements analysts have finished developing the specification. Some effort and cost models use parameters based on past experience to apportion the effort across the software development life cycle. For instance, the original COCOMO model suggested effort required by development activities, based on percentages allotted to key process activities. But, as Figure 3.14 illustrates, researchers...
3.4 Risk Management

As we have seen, many software project managers take steps to ensure that their projects are done on time and within budget and cost constraints. However, project managers must determine which risks are most important to manage and how to assess and control the risks on their projects.

What is a Risk?

Many events occur during software development. Sidebar 3.4 lists Bohm's view of some of the risks that we should consider. We should identify the risks and threats that we’ll encounter.

Risk can be defined as the event that will occur. To evaluate a project, we must have some idea of the probability that the event will occur, the consequences, and the response. To evaluate a project, we must have some idea of the probability that the event will occur, the consequences, and the response. To evaluate a project, we must have some idea of the probability that the event will occur, the consequences, and the response.

1. A less associated with the event. The event must occur at some future point in time. The event must occur at some future point in time. The event must occur at some future point in time.

2. The likelihood of the event will occur. We must have some idea of the probability of the event. We must have some idea of the probability of the event.

3. The degree of impact or severity that the event will occur. We must have some idea of the probability of the event. We must have some idea of the probability of the event.

4. Risk control or avoidance. We can reduce the impact of the event. Risk control or avoidance. We can reduce the impact of the event.

5. Risk management involves several important aspects, each of which is illustrated in Figure 5.1. First, you assess the risks on your project, then you act on what you’ve identified.
during the course of development or maintenance. The assessment consists of three main steps: identify the risks, analyze the impact and probability of each risk, and prioritize them to control the risks. To identify these risks, you may use any of the following techniques:

- The system under consideration is a complex system. To identify risks, you may use the checklist technique, which involves identifying a list of potential risks that may be common to the system. You may be able to anticipate problems that may occur in the system by identifying the risks involved.

- The system under consideration is a complex system. To identify risks, you may use the checklist technique, which involves identifying a list of potential risks that may be common to the system. You may be able to anticipate problems that may occur in the system by identifying the risks involved.

- The system under consideration is a complex system. To identify risks, you may use the checklist technique, which involves identifying a list of potential risks that may be common to the system. You may be able to anticipate problems that may occur in the system by identifying the risks involved.

Finally, you analyze the risks you have identified so you can understand the possible impact and where they might occur. Here are some techniques you can use to enhance your understanding:

- Include system dynamics models, system models, production models, network analysis, and more.

- The system under consideration is a complex system. To identify risks, you may use the checklist technique, which involves identifying a list of potential risks that may be common to the system. You may be able to anticipate problems that may occur in the system by identifying the risks involved.

- The system under consideration is a complex system. To identify risks, you may use the checklist technique, which involves identifying a list of potential risks that may be common to the system. You may be able to anticipate problems that may occur in the system by identifying the risks involved.

The risk exposure is computed from the risk impact and the risk probability, so you must estimate each of these risk aspects. To see how the quantification is done, consider the analysis depicted in Figure 3. The Suppose case refers to the system under development process and consider your overall underlined risks. In this case, there is a one-in-one-hundred chance of failure, which could be considered as a low probability. The risk exposure is the product of the risk impact and probability. In this case, the risk exposure is equal to the product of the risk impact and probability, which is equal to one-in-one-hundred. Therefore, the risk exposure is low. For example, if the risk impact is 10 and the probability is 0.01, the risk exposure is 0.1, which is considered as a low risk.
control acknowledges that we may not be able to identify all risks. Instead, we may be able to minimize the risk or mitigate it by taking action to handle the unidentified outcomes in an acceptable way. Therefore, risk control involves risk reduction, risk planning, and risk resolution.

There are three strategies for risk reduction:

- avoiding the risk by changing requirements, performing a feasibility study, or performing a risk analysis to determine the level of risk.
- assigning the risk to another person or business in such a way that the company becomes riskier.
- assuming the risk by accepting it and committing to it with the project's resources.

To aid decision making about risk reduction, we must take into account the cost of reducing the risk. We can evaluate the difference in risk exposure by dividing the cost of reducing the risk by the level of reduction risk.

(Risk exposure before reduction - risk exposure after reduction) / cost of reducing risk

If the cost of reduction is not high enough to justify the action, then we can look for other less costly or more effective reduction techniques.

In some cases, we can choose a development process to help reduce the risk, for example, we can use the Model in Chapter 2 that provides an improved definition of the requirements and design and a prototyping process can reduce many project risks. It is helpful to record your decision in a risk management plan so that both customers and development team can review them, as well as, as they get to be habit, they get used to it. They should monitor the project development progress, periodically re-evaluating the project. they, their responsibilities, and their likely impact.

3.5 THE PROJECT PLAN

To communicate risk, analysis, and management, project cost estimates, schedule, and organization in our customers, we usually write a document called a project plan. The plan puts in writing the customer's needs, as well as, as what we hope to do to meet them. The customer can refer to the plan for information about activities in the development process, making it easy to follow the project's progress. As we develop this plan, we can use it to confirm the customer's assumptions, as we make especially about cost and schedule.

A good project plan includes the following items:

1. project scope
2. project schedule
3. project team organization
4. technical description of the proposed system
5. project standards, procedures, and proposed techniques and tools
6. quality assurance plan
7. configuration management plan
8. documentation plan
9. data management plan
10. source management plan
11. test plan
12. training plan
13. security plan
14. risk management plan
15. maintenance plan

The scope defines the system boundary, explaining what will be included in the system and what will not be included. As we discuss the customer's needs, we understand what is important.

The schedule can be created using a work breakdown structure, the deliverables, and a timeline to show what will be happening at each point during the project life cycle. A Gantt chart can be useful in illustrating the parallel nature of some of the development tasks.

The project plan also has the people on the development team that they are organized and what they will be doing. As we work, we continuously need to remain on time during the project, as the plan usually contains a resource allocation chart that shows staffing levels at different times.

Writing a technical description for our customers answers questions and addresses issues as we anticipate how development will proceed. This description lists hardware and software, including computers, software, and special-purpose equipment or software. Any special restrictions on eating, sleeping, or other constraints that are important or are documented in the plan. The plan also lists any standards, methods, or tools that must be used, such as:

- algorithms
- tools
- review or inspection techniques
- design languages or representations
- coding conventions
- testing techniques

For large projects, it may be appropriate to include a separate quality assurance plan to describe how reviews, inspections, testing, and other techniques will help to evaluate quality and assure that it meets the customer's needs. Similarly, large projects need a configuration management plan, especially when there are multiple versions and releases of the system. As we see in Chapter 10, configuration management helps to control the correct version of the software. The configuration management plan will describe how we will track changes of the requirements, design, code, test plans, and documents.

Many documents are produced during development, especially for large projects where information about the design must be made available to teams and will be maintained, including source code, and components. The project plan lists the documents that will be produced, when they will be written, and who will write them and who will make them and, in contact with the configuration management plan, describes how documents will be changed.
Because every software system involves data for input, calculation, and output, the project plan must explain how data will be gathered, stored, manipulated, and archived. The plan should also explain how resources will be used. For example, if the hardware configuration includes removable disks, then the resource management part of the project plan should explain what data are on each disk and how the disk packs or diskettes will be allocated and backed up.

Testing requires a great deal of planning to be effective, and the project plan describes the project's overall approach to testing. In particular, the plan should state how test data will be generated, how the program module will be tested (e.g., by testing all paths or all statements), how program modules will be integrated with each other and tested, how the entire system will be tested, and who will perform each type of testing. Sometimes, systems are produced in stages or phases, and the test plan should explain how each stage will be tested. When new functionality is added to a system in stages, as we saw in Chapter 2, the test plan must address regression testing, ensuring that the existing functionality still works correctly.

Training classes and documents are usually prepared during development, rather than after the system is complete, so that training can begin as soon as the system is ready and sometimes before. The project plan explains how training will occur, describing each class, supporting software and documents, and the expertise needed by each student.

When a system has security requirements, a separate security plan is sometimes needed. The security plan addresses the way that the system will protect data, users, and hardware. Since security involves confidentiality, integrity, and availability, three security must explain how each factor affects system development. For example, if access to the system is limited by using passwords, then the plan must describe who issues and maintains the passwords, who develops the password-handling software, and what the password encryption scheme will be.

Finally, if the project team will maintain the system after it is delivered to the user, the project plan should describe responsibilities for changing the code, repairing the hardware, and updating supporting documentation and training materials.

3.6 PROCESS MODELS AND PROJECT MANAGEMENT

We have seen how different aspects of a project can affect the effort, cost, and schedule, as well as the risks involved. Managers must successfully build quality products on time and within budget; see those who take the project management techniques to the particular characteristics of the resources needed, the chosen process, and the people assigned.

To understand what to do on your next project, it is useful to examine project management techniques used by successful projects from the recent past. In this section, we look at two projects: Digital's Alpha AXP program and the F-16 aircraft software. We also investigate the merging of process and project management.

Enrollment Management

Digital Equipment Corporation spent many years developing its Alpha AXP system, a new system architecture and associated products that formed the biggest project in the company's history. The software portion of the effort involved four operating systems and 22 software engineering groups whose roles included designing migration tools, network software, and decision-making support applications for other development projects. The major problems were with the Alphas involved, as well as with the management of the project. During the course of development, the project managers developed a model that incorporated four elements called the Enrollment Management model:

1. establishing an appropriate role and role model
2. developing a high-purpose plan
3. insuring a high-purpose plan ensures that commitments are made from participants
4. acknowledging every advance and learning in the program proposed (Cookin 1990)

Figure 3.17 illustrates the model. The x-axis was used to "scour" the related programs, so that all addressed common goals. Each group or subgroup of the project defined its own goals. Next, as managers developed plans, they delegated tasks to groups, soliciting input from each. Each required task was measurable and identified with a particular manager or person responsible for getting the work done on time. Project team members were asked to identify these next steps, and when a step threatened to keep them from meeting its commitments, the project manager declared the project to be a "critical" event. Such a declaration meant that team members were expected to make substantial changes to help move the project forward. For each project goal, what had been learned, and how things could be improved were discussed.

![FIGURE 3.17] Enrollment Management model (Cookin, 1990)
Coordinating all the hardware and software groups was difficult, and managers realized that they had to oversee both technical and project events. That is, the technical focus involved technical design and strategy, whereas the project focus addressed commitments and deliverables. Figure 3.18 illustrates the organization that allowed both facets to contribute to the overall program.

The simplicity of the model and organization does not mean that managing the Alpha program was simple. Several workshops were organized, and the program was designed to create an overall plan with well-defined project milestones and specifications for each task. The work was divided among different groups, and each group was responsible for a specific part of the project. This approach allowed for better coordination and control of the overall project.

Accountability Modeling

The US Air Force and Lockheed Martin formed an Integrated Product Development Team to build a modular software system designed to increase capacity and decrease time and cost of future software changes to the F-16. The project was structured as a separate organization, with the primary goal of minimizing the cost and time required for future software changes. The team was responsible for developing a modular system that could be easily modified and updated in the future.

The project was broken down into smaller tasks, and each task was assigned to a specific group. This approach allowed for better accountability and control of each task. The project managers were responsible for overseeing the progress of each task and ensuring that the project was on track.

The project was successful, and the team was able to deliver a software system that met the requirements of the F-16. The system was modular and easily modified, which allowed for future changes to be made without disrupting the overall system. The project was completed on time and within budget, which was a significant accomplishment given the complexity of the task.

The success of the project was due to the effective use of accountability modeling. The team was able to break down the project into smaller tasks and assign each task to a specific group. This allowed for better accountability and control of each task, which ultimately led to the successful completion of the project.
sense for both the team and the stakeholders. The model was applied to the design of management systems and to team operating procedures, replacing independent behaviors with interdependence, emphasizing "being good rather than looking good" (Parris 1996).

As a result, several practices were required, including a weekly, one-hour team status review. To reinforce the notions of responsibility and accountability, each personal action item had explicit closure criteria and was recorded to completion. An action item could be assigned to a team member or a stakeholder, and often involved clarifying issues or requirements, providing missing information or resolving conflicts.

Because the team had multiple, overlapping activities, an activity map was used to illustrate progress on each activity in the overall context of the project. Figure 3.20 shows part of an activity map. You can see how each bar represents an activity, and each activity is assigned a method for reporting progress. The point on a bar indicates when detailed planning should be in place to guide activities. The 'today' line shows current status, and an activity map was used during the weekly reviews as an overview of the progress to be discussed.

For each activity, progress was tracked using an appropriate evaluation or performance method. Sometimes the method included cost estimation, critical path analysis, or scheduled tracking. Earned value was used as a common measure for comparing progress on different activities. A scheme for comparing activities determined how much of the project had been completed by each activity. The earned value calculation included weights to represent what percent of the total process each step contributed, relative to overall effort. Similarly, each component was assigned a unit value that represented its proportion of the total product, so that progress relative to the final size could be tracked. Then, an earned value summary chart similar to Figure 3.21 was presented at each review meeting. Once part of a product was completed, its progress was no longer tracked. Instead, its performance was tracked, and problems were recorded. Each problem was assigned...
A priority by the stakeholders, and a snapshot of the top five problems on each product team's list was presented at the weekly meeting for discussion. The priority list generated discussion about why the problems occurred, what work around could be put in place, and how similar problems could be prevented in the future.

The project managers found a major problem with the accountability model. It took them nothing about coordination among different teams. As a result, they built software to track the handoffs from one team to another, so that every team could understand who was waiting for action on projects from another team. A model of the handoff was used for planning, so that unchangeable patterns or conditions could be eliminated. Thus, an examination of the handoff model became part of the project review.

It is easy to see how the accountability model, coupled with the handoff model, addressed several aspects of project management. First, it provided a mechanism for communication and coordination. Second, it ensured risk management, especially by forcing these teams to examine problems in review meetings. Third, the model facilitated progress reporting and process¹ solving. Thus, the model actually describes a project management process that was followed on the 1-16 project.

Anchor Milestones

In Chapter 2, we examined many process models that described how the technical activities of software development should proceed. Then, in this chapter, we looked at several methods to organize people to perform those activities. The Alpha AX and F-16 examples have shown us that project management must be tightly integrated with the development process, not just for tracking progress, but more importantly for effective planning and decision making to prevent major problems from occurring the project. Boehm (1996) has identified three milestones common to all software development processes that can serve as a basis for both technical process and project management:

- Life cycle objectives
- Life cycle architecture
- Initial operational capability

We can examine each milestone in more detail.

The purpose of the life cycle objectives milestone is to make sure that the stakeholders agree with the system's goals. The key stakeholders are a team to determine the system boundary, the environment in which the system will operate, and the external systems with which the system must interact. Then, the stakeholders work through scenarios of how the system will be used. The scenarios can be expressed in terms of prototypes, sequent layouts, data flows, or other representations, some of which we will learn about in later chapters. If the system is mission-critical or safety critical, the scenarios should also include instances where the system fails so that designers can determine how the system is supposed to react or even avoid a critical failure. Similarly, other essential features of the system are derived and agreed upon. The result is an initial life cycle plan that lays out (Boehm 1996):

- Objectives: Why is the system being developed?
- Milestones and schedules: What will be done by whom?
- Responsibilities: Who is responsible for a function?
- Approach: How will the job be done, technically and organizationally?

Section 3.6 Process Models and Project Management

- Objectives: How much of each resource is needed?
- Feasibility: Can this be done, and is there a good business reason for doing it?

The life cycle architecture is coordinated with the life cycle objectives. The purpose of the life cycle architecture milestone is to define both the system and the software architecture, the components of which we will study in Chapter 5.6 and 7. The architectural choices must address the project risks addressed by the risk management plan, focusing on system evolution in the long term as well as system requirements in the short term.

The key elements of the initial operational capability are the readiness of the software itself, the site in which the system will be used, and the selection and training of the team that will use it. Boehm notes that different processes can be used to implement the initial operational capability, and different estimating techniques can be applied at different stages.

Supplement these milestones, Boehm suggests using the Win-Win spiral model, illustrated in Figure 3.22 and intended to be an extension of the spiral model we examined in Chapter 2. The model encourages participants to solve the problem, understand the system's next-level alternatives, and constraints.

Boehm applied Win-Win, called the Theory W approach, to the U.S. Department of Defense's STARs program, where focus was on developing a set of prototype software engineering environments. The Win-Win approach was a good candidate for Theory W, because there was a great mismatch between what the government was planning to build and what the potential users needed and wanted. The Win-Win model led to several key compromises, including negotiation of a set of common, open interface specification for development to reach a larger market share at lower costs, and the inclusion of several demonstration projects to reduce risk. Boehm reports that as of 1999, costs on the project were reduced from $140 to $57 per delivered line of code, and that quality improved from 3 to 0.035 fault per thousand delivered lines of code. Several other projects reported similar successes. Boehm developed over half a software baseline for complex distributed software within budget and schedule using Boehm's milestones with live increments. The first increment included distributed kernel software as part of the life cycle architecture, milestones; the project was required to demonstrate its ability to meet projections that the number of requirements would grow over time (Boehm 1996).

FIGURE 3.22 Win-Win spiral model (Boehm, 1996).


### Table 3.10: Scale Factors for COMCOS II Early-Design-Prediction Models

<table>
<thead>
<tr>
<th>Scale Factor</th>
<th>Very Low</th>
<th>Low</th>
<th>Normal</th>
<th>High</th>
<th>Very High</th>
<th>Extremely High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>largely unknown</td>
<td>moderately</td>
<td>somewhat known</td>
<td>generally familiar</td>
<td>familiar</td>
<td>thoroughly familiar</td>
</tr>
<tr>
<td>Flexibility</td>
<td>tightness</td>
<td>occasional</td>
<td>frequent</td>
<td>general</td>
<td>common</td>
<td>rare</td>
</tr>
<tr>
<td>Versatility</td>
<td>little experienced</td>
<td>some</td>
<td>frequently</td>
<td>generally</td>
<td>mostly</td>
<td>extremely</td>
</tr>
<tr>
<td>Experience</td>
<td>very different</td>
<td>most difficult</td>
<td>significantly</td>
<td>largely</td>
<td>highly</td>
<td>almost</td>
</tr>
<tr>
<td>Processability</td>
<td>determined by experience</td>
<td>determined by expectations</td>
<td>determined by process</td>
<td>determined by technology</td>
<td>determined by research</td>
<td>determined by application</td>
</tr>
</tbody>
</table>

---

We add all the weights in the rightmost column to generate a curve of new application points (NAPs). Suppose our developers have low experience and low CASE maturity. Table 3.12 tells us that the productivity rate for this circumstance is 7. Then the COMCOS model tells us that the estimated effort to build the Piccadilly system is NOT divided by the productivity rate, or 729 person-months.

As we understand more about the requirements for Piccadilly, we can use the other parts of COMCOS: the early design model and the prototyping model, based on normal effort estimates derived from lines of code or function points. These models use a scale factor computed from the project's scale factors, listed in Table 3.16.

*Extra high* is equivalent to a rating of zero, *very high* to 1, *high* to 2, *normal* to 3, *low* to 4, and *very low* to 5. Each of the scale factors is multiplied by the sum of all weights to get an initial effort estimate. For example, suppose we know that the type of application we are building for Piccadilly is generally familiar to the development team. We can rate the first scale factor as *high*. Similarly, we may rate the second scale factor as *very high*. All together, the initial effort estimate is then 19. We can then adjust the factor by the number of months. For example, suppose we multiply by 182 months to get an adjusted effort estimate of 3500 person-months.
understanding the planning process and estimation techniques gives you a good idea of how your input will be used to make decisions for the whole team. Also, we have seen how the number of possible communication paths grows as the size of the team increases. You can take communication into account when you are planning your work and estimating the time it will take you to complete your next task.

We have also seen how communication styles affect and how they affect the way we interact with others on the job. By understanding your teammates’ styles, you can create reports and presentations for them that match their expectations and needs. You can prepare summary information for people with a bottom-line style and others complete analytical information to those who are rational.

10 WHAT THIS CHAPTER MEANS FOR YOUR DEVELOPMENT TEAM

At the same time, you have learned how to organize a development team so that team interaction helps produce a better product. There are several choices for team structure, from a hiearchical chain of programmer teams to a loose, egalitarian approach. Each has its benefits and each depends to some degree on the uncertainty and size of the project.

We have also seen how the team can work to anticipate and resolve the project’s beginning. Redundancy, functionality, team reviews, and other techniques can help us catch errors early before they become embedded in the code as faults waiting to cause failures.

Similarly, cost estimation should be done early and often, including inputs from team members about progress in specifying, designing, coding, and testing the system. Cost estimation and risk management can work hand in hand as cost estimates raise concerns about finishing on time and within budget. Risk management techniques are used to mitigate or even eliminate risks.

11 WHAT THIS CHAPTER MEANS FOR RESEARCHERS

This chapter has described many techniques that still require a great deal of research. Little is known about which team organizations work best in which situations. Likewise, cost- and schedule-estimation models are not as accurate as we would like them to be, and improvements can be made to learn more about how project teams, processes, and resources change and affect efficiency and productivity. Some methods, such as machine learning, look promising but require a great deal of historical data to make them accurate. Researchers can help us understand how to balance practicality with accuracy when using estimation techniques.

Similarly, a great deal of research is needed in making risk management techniques practical. The calculation of risk exposure is currently more art than science and we need methods to help us make our risk calculations more relevant and our mitigation techniques more effective.

TERM PROJECT

Often, a company or organization must estimate the effort and time required to complete a project, even before detailed requirements are defined. Using the approaches described in this chapter, or a tool of your choosing from other sources, estimate the...
KEY REFERENCES

A great deal of information about COCOMO is available from the Center for Software Engineering at the University of Southern California. The Web site (http://www.cs.usc.edu/cocomo) contains a tutorial and a discussion of COCOMO, including a few implementations of COCOMO. It is this site that you can also find out about COCOMO and the concepts and techniques used in the COCOMO II model. The COCOMO II model uses more data and information about the specific project that is being estimated. It is available from IFIP, the International Federation for Information Processing.

The Center for Software Engineering also performs research in risk management. You can read more about COCOMO at the center's Web site (http://www.cs.usc.edu/cocomo).

The Center for Software Engineering also performs research in risk management. You can read more about COCOMO at the center's Web site (http://www.cs.usc.edu/cocomo).

4 Capturing the Requirements

In this chapter we look at:

- eliciting requirements from our customers
- types of requirements
- notations and methods for capturing requirements
- reviewing requirements to ensure their quality
- documenting requirements for use by the design and test teams

In earlier chapters we discussed the stages of system development. When looking at various process models we noted several key steps for successful software development. Each proposed model of the software development process includes activities aimed at capturing requirements—understanding what the customers and users expect the system to do. Thus, the systems engineer and system analyst must gain an understanding of what the customer requires. In this chapter, we will look at how to capture the requirements as a set of functional and nonfunctional constraints. We will explore the characteristics of each and then discuss the various notations and methods for capturing requirements. As we shall see, we must find requirements on which both we and the customer agree and with which we can build our test procedures. First, let us determine exactly what is required and how we work with users to define and document it.