Project planning is one of the most important jobs of a software project manager. As a manager, you have to break down the work into parts and assign those to project team members, anticipate problems that might arise, and prepare tentative solutions to these problems. The project plan, which is created at the start of a project, is used to communicate how the work will be done to the project team and customers, and to help assess progress on the project.

Project planning takes place at three stages in a project life cycle:

1. At the proposal stage, when you are bidding for a contract to develop or provide a software system. You need a plan at this stage to help you decide if you have the resources to complete the work and to work out the price that you should quote to a customer.

2. During the project startup phase, when you have to plan how work will be done on the project, how the project will be broken down into increments, how resources will be allocated across your company, etc. Here, you have more information than at the proposal stage, and can therefore refine the initial effort estimates that you have prepared.

3. Periodically throughout the project, when you modify your plan in light of experience gained and information from monitoring the progress of the work. You learn more about the system being implemented and capabilities of your development team. This information allows you to make more accurate estimates of how long the work will take. Furthermore, the software requirements are likely to change in such a manner that the work breakdowns have to be altered and the schedule extended. For traditional development projects, this means that the plans created during the startup phase have to be modified. However, when an agile approach is used, plans are shorter term and continually change as the software evolves. I discuss agile planning in Section 23.4.

Planning at the proposal stage is inevitably speculative, as you do not usually have a complete set of requirements for the software to be developed. Rather, you have to respond to a call for proposals based on a high-level description of the software functionality that is required. A plan is often a required part of a proposal, so you have to produce a credible plan for carrying out the work. If you win the contract, you then usually have to replan the project, taking into account changes since the proposal was made.

When you are bidding for a contract, you have to work out the price that you will propose to the customer for developing the software. As a starting point for calculating this price, you need to draw up an estimate of your costs for completing the project work. Estimation involves working out how much effort is required to complete each activity and, from this, calculating the total cost of activities. You should always calculate software costs objectively, with the aim of accurately predicting the cost of developing the software. Once you have a reasonable estimate of the likely costs, you are then in a position to calculate the price that you will quote to the customer. As I discuss in the next section, many factors influence the pricing of a software project—if it is not simply cost + profit.

There are three main parameters that you should use when comparing the costs of a software development project:

- effort costs (the costs of paying software engineers and managers);
- hardware and software costs, including maintenance;
- travel and training costs.

For most projects, the biggest cost is the effort cost. You have to estimate the total effort (in person-months) that is likely to be required to complete the work of a project. Obviously, you have limited information to make such an estimate, so you have to make the best possible estimate and then add significant contingency (extra time and effort) in case your initial estimate is optimistic.

For commercial systems, you normally use commodity hardware, which is relatively cheap. However, software costs can be significant if you have to license middleware and platform software. Extensive travel may be needed when a project is developed at different sites. Although travel costs themselves are usually a small fraction of the effort costs, the time spent traveling is often wasted and adds significantly to the effort costs of the project. Electronic meeting systems and other software that supports remote collaboration can reduce the amount of travel required. The time saved can be devoted to more productive project work.

Once a contract to develop a system has been awarded, the outline project plan for the project has to be refined to create a project startup plan. At this stage, you should know more about the requirements for this system. However, you may not have a complete requirements specification, especially if you are using an agile approach to development. Your aims at this stage should be to create a project plan that can be used to support decision making about project staffing and budgeting. You use the plan as a basis for allocating resources to the project from within the organization and to help decide if you need to hire new staff.

The plan should also define project monitoring mechanisms. You must keep track of the progress of the project and compare actual and planned progress and costs. Although most organizations have formal procedures for monitoring a project, the manager should be able to form a clear picture of what is going on through informal discussions with project staff. Informal monitoring can predict potential project problems by revealing difficulties as they occur. For example, daily discussions with project staff might reveal a particular problem in finding a software fault. Rather than waiting for a schedule slip-up to be reported, the project manager could then immediately assign more expert to the problem, or decide to program around it.

The project plan always evolves during the development process. Development planning is intended to ensure that the project plan remains a useful document for staff to understand what is to be achieved and when it is to be delivered. Therefore, the schedule, cost estimate, and risks all have to be revised as the software is developed.

If an agile method is used, there is still a need for a project startup plan, although the approach used, the company still needs to plan how resources will be allocated to a project. However, this is not a detailed plan and should include only limited information about the work breakdown and project schedule. During development, an informal project plan and effort estimates are drawn up for each release of the software, with the whole team involved in the planning process.

### 23.2 Software pricing

In principle, the price of a software product to a customer is simply the cost of development plus profit for the developer. In practice, however, the relationship between the project cost and the price quoted to the customer is not usually as simple. When calculating a price, you should take broader organizational, economic, political, and business considerations into account, such as those shown in Figure 23.1. You need to think about organizational concerns, the risks associated with the project, and the type of contract that will be used. These may cast the price to be adjusted upwards or downwards. Because of the organizational considerations involved, deciding on a project price should be a group activity involving marketing and sales staff, senior management, and project managers.

To illustrate some of the project pricing issues, consider the following scenario:

A small software company, PharmaSoft, employs 10 software engineers. It has just finished a large project but only has contracts in place that require five development staff. However, it is bidding for a very large contract with a major pharmaceutical company that requires 30 person-years of effort over two years. The project will not start for at least 12 months but, if granted, it will transform the finances of the company.

PharmaSoft gets an opportunity to bid on a project that requires six people and has to be completed in 10 months. The costs (including overheads of this project) are estimated at $1.2 million. However, to improve its competitiveness...
23.2 Plan-driven development

Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail. A project plan is created that records the work to be done, who will do it, the development schedule, and the work products. Managers use the plan to support project decision making and as a way of measuring progress. Plan-driven development is based on engineering project management techniques and can be thought of as the "traditional" way of managing large software development projects. This contrasts with agile development, where many decisions affecting the development are delayed and made later, as required, during the development process.

The principal argument against plan-driven development is that many early decisions have to be made because of changes to the environment in which the software is to be developed and used. Delaying such decisions is sensible because it avoids unnecessary rework. The arguments in favor of a plan-driven approach are that early planning allows organizational issues (such as the availability of staff, other projects, etc.) to be closely taken into account, and that potential problems and dependencies are discovered before the project starts, rather than once the project is under way.

In my view, the best approach to project planning involves a judicious mixture of plan-based and agile development. The balance depends on the type of project and the skills of the people who are available. At one extreme, large, secure and safety-critical systems require extensive up-front analysis and may have to be certified before they are put into use. These should be mostly plan-driven. At the other extreme, small, medium-size information systems, to be used in a rapidly changing competitive environment, should be mostly agile. Where several companies are involved in a development project, a plan-driven approach is normally used to coordinate the work across each development site.

23.2.1 Project plans

In a plan-driven development project, a project plan sets out the resources available to the project, the work breakdown, and a schedule for carrying out the work. The plan should identify risks to the project and the software under development, and the approach that is taken to risk management. Although the specific details of project plans vary depending on the type of project and organizational plans normally include the following sections:

1. Introduction: This briefly describes the objectives of the project and sets out the

![system plan process]

Figure 23.3 The project planning process

shows a typical workflow for a project planning process. Plan changes are inevitable. As more information about the system and the project team becomes available during the project, you should regularly review the plan to reflect requirements, schedule, and risk changes. Changing business goals also leads to changes in project plans. As business goals change, this can affect all projects, which may then have to be replanned.

At the beginning of a planning process, you should assume the constraints affecting the project. These constraints are the required delivery date, staff available, overall budget, available tools, and so on. In conjunction with this, you should also identify the project milestones and deliverables. Milestones are points in the schedule against which you can assess progress, for example, the handover of the system for testing. Deliverables are work products that are delivered to the customer, such as requirements documents for the system.

The process then enters a loop. You draw up an estimated schedule for the project and the activities defined in the schedule are initiated or given permission to continue. After some time (usually about two to three weeks), you should review progress and note discrepancies from the planned schedule. Because initial estimates of project parameters are inevitably approximate, minor slippages are normal and you will have to make modifications to the original plan.

It is important to be realistic when you are creating a project plan. Problems of some degree are nearly always arise during a project, and these can lead to project delays. Your initial assumptions and scheduling should therefore be pessimistic rather than optimistic. There should be sufficient contingency built into your plan so that the project committee and milestones don’t need to be renegotiated every time you go around the planning loop.

If there are serious problems with the development work that are likely to lead to significant delays, you need to initiate risk mitigation actions to reduce the risks of

In summary, you should not rely on the plan and should plan to change the plan as the project progresses.

23.2.2 The planning process

Project planning is an iterative process that starts when you create an initial project when diving the project into smaller chunks. 

![milestones and deliverables]

Figure 23.1 Factors affecting software pricing

This means that, although a loss makes money on this contract, it can retain specialist staff for the more profitable future projects that are likely to come on stream in a year’s time.

As the cost of the project is only loosely related to the price quoted a customer, ‘pricing to win’ is a commonly used strategy. Pricing to win means that a company has some idea of the price that the customer expects to pay and makes a bid for the contract based on the customer’s perceived price. This may seem unethical and uncompetitive, but it does have advantages for both the customer and the system provider.

A project cost is agreed on the basis of an outline proposal. Negotiations then take place between client and customer to establish the detailed project specification. This specification is constrained by the agreed cost. The buyer and seller must agree on what is acceptable system functionality. The fixed factor in many projects is the effort required but the cost of the requirements may be changed so that the cost is not essential.

For example, say a company (GoSilk) is bidding for a contract to develop a fuel delivery system for an oil company that schedules deliveries of fuel to its service stations. There is no detailed requirements document for this system, so GoSilk estimates that a price of $500,000 is likely to be competitive and within the oil company’s budget. After they are granted the contract, GoSilk then negotiates the detailed requirements of the system so that the basic functionality is delivered. They then estimate the additional costs for other requirements. The oil company does not
This may involve renegotiating the project constraints and deliverables with the customer. A new schedule of when work should be completed also has to be established and agreed upon with the customer.

If this renegotiation is unsuccessful or the risk mitigation actions are ineffective, then you should arrange for a formal project technical review. The objectives of this review are to find an alternative approach that will allow the project to continue, and to check whether the project and the goals of the customer and software developer are still aligned.

The outcome of a review may be a decision to cancel a project. This may be a result of technical or managerial failures but, more often, it is a consequence of external changes that affect the project. The development time for a large software project is often several years. During this time, the business objectives and inevitably change. These changes may mean that the software is no longer required or that the original project requirements are inappropriate. Management may then decide to stop software development or to make major changes to the project to reflect the changes in the organizational objectives.

23.3 Project scheduling

Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed. You estimate the calendar time needed to complete each task, the effort required, and who will work on the tasks that have been identified. You also have to estimate the resources needed to complete each task, such as the disk space required on a server, the time required on specialized hardware, such as a simulator, and what the total budget will be. In terms of the planning stages that I discussed in the introduction of this chapter, an initial project schedule is usually created during the project startup phase. This schedule is then refined and modified during development planning.

Both plan-based and agile processes need an initial project schedule, although the level of detail may be less in an agile project plan. This initial schedule is used to plan how people will be allocated to projects and to check the progress of the project against its contractual commitments. In traditional development processes, the complete schedule is initially developed and then modified as the project progresses. In agile processes, there is to be an overall schedule that identifies when the major phases of the project will be completed. As an iterative approach to scheduling is then used to plan each phase.

Scheduling in plan-driven projects (Figure 23.4) involves breaking down the total work involved in a project into separate tasks and estimating the time required to complete each task. Tasks should normally last at least a week, and no longer than 2 months. Short subdivision means that a disproportionate amount of time must be spent on coordination and coordination of the effort. The maximum amount of time that an activity can be delayed is limited and the project could still be completed within the time constraints. This is known as the critical path of the project.

Figure 23.4: The project scheduling process

23.3.1 Schedule representation

Project schedules may simply be represented in a table or spreadsheet showing the tasks, effort, expected duration, and task dependencies (Figure 23.5). However, this required are specified in the project contract and the customer's view of the project's progress depends on these deliverables.

To illustrate the chart and I have created a hypothetical set of tasks as shown in Figure 23.5. This table shows tasks, estimated effort, duration, and task dependencies. From Figure 23.5, you can see that task T1 is dependent on task T1. Task T1 must, therefore, be completed before T2 starts. For example, T1 might be the preparation of a component design and T3, the implementation of that design. Before implementation starts, the design should be complete. Notice that the estimated duration for some tasks is more than the effort required and vice versa. If the effort is less than the duration of the task, this means that the people allocated to that task are not working full-time on it. If the effort exceeds the duration, this means that several team members are working on the task at the same time.

Figure 23.5 shows the dependency information in a graphical format. It is a bar chart showing a project calendar and the start and finish dates of tasks. Reading from left to right, the bar chart clearly shows when tasks start and end. The milestones (M1, M2, etc.) are also shown on the bar chart. Notice that tasks that are dependents are carried out in parallel (e.g., tasks T1, T2, and T4 all start at the beginning of the project).

As well as planning the delivery schedule for the software, project managers have to allocate resources to tasks. The key resource, of course, is the software engineers who will do the work, and they have to be assigned to project activities. The resource allocation can also be input to project management tools and a bar chart generated. This shows how various tasks will be carried out over time.
working on more than one task at the same time and, sometimes, they are not working on the project. They may be on holiday, working on other projects, attending training courses, or engaging in some other activity. I show part-time assignments using a diagonal line crossing the bar.

Large organizations usually employ a number of specialists who work on a project when needed. In Figure 23.7, you can see that Mary is a specialist who works on only a single task in the project. This can cause scheduling problems. If one project is delayed while a specialist is working on it, this may have a knock-on effect on other projects where the specialist is also required. These may then be delayed because the specialist is not available.

If a task is delayed, this can obviously affect later tasks that are dependent on it. They cannot start until the delayed task is completed. Delays can cause serious problems with staff allocation, especially when people are working on several projects at the same time. If a task (T) is delayed, the people allocated to it may be assigned to other work (V). To complete this, they may take longer than the delay but, once assigned, they cannot simply be reassigned back to the original task. T. This may then lead to further delays in T or other rear entries.

Figure 23.6 Activity bar chart

I have already discussed the Scrum approach to planning in Chapter 2, so I concentrate here on planning in extreme programming (XP). This is the 'planning game' and it usually involves the whole development team, including customer representatives. Figure 23.8 shows the stages in the planning game.

The system specification in XP is based on a story board that reflects the features that should be included in the system. At the start of the project, the team and the customer try to identify a set of stories, which cover all of the functionality that will be included in the final system. Some functionality will inevitably be missing, but this is not important at this stage.

The next stage is an estimation stage. The project team reads and discusses the stories and ranks them in order of the amount of time they think it will take to implement the story. This may involve breaking large stories into smaller stories. Relative estimation is often easier than absolute estimation. People often find it difficult to estimate how much effort or time is needed to do something. However, when people are presented with several things to do, they can make judgments about which stories will take the longest time and most effort. Once the ranking has been completed, the team allocates notional effort points to the stories. A complex story may have 5 points and a simple story 2 points. You do this for all of the stories in the ranked list.

Once the stories have been estimated, the relative effort is translated into the first estimate of the total effort required by using the notion of "velocity". In XP, velocity is the number of effort points implemented by the team, per day. This can be estimated either from previous experience or by developing one or two stories to see how much time is required. The velocity estimate is approximate, but is refined during the development process. Once you have a velocity estimate, you can calculate the total effort in person-days to implement the system.

Release planning involves selecting and refining the stories that will be implemented in a release of a system and the order in which the stories should be implemented. The customer has to be consulted on developing one or two stories to see how much time is required. The velocity estimate is approximate, but is refined during the development process. Once you have a velocity estimate, you can calculate the total effort in person-days to implement the system.

Release planning involves selecting and refining the stories that will be implemented in a release of a system and the order in which the stories should be implemented. The customer has to be consulted on developing one or two stories to see how much time is required. The velocity estimate is approximate, but is refined during the development process. Once you have a velocity estimate, you can calculate the total effort in person-days to implement the system.
that it is impossible to estimate system development costs accurately during the early stages of a project.

There is even a fundamental difficulty in assessing the accuracy of different approaches to cost and effort estimation. Project estimates are often self-fulfilling. The estimate is used to define the project budget and the product is adjusted so that the budget figure is realized. A project that is within budget may have achieved this at the expense of features in the software being developed.

I do not know of any controlled experiments with project costing where the estimated costs were not used to bias the experiment. A controlled experiment would not reveal the cost estimate to the project manager. The actual costs would be compared with the estimated project costs. Nevertheless, organizations need to make software effort and cost estimates. There are two types of technique that can be used to do this:

1. Experience-based techniques. The estimate of future effort requirements is based on the manager’s experience of past projects and the application domain. Essentially, the manager makes an informed judgment of what the effort requirements are likely to be.

2. Algorithmic cost modeling. In this approach, a mathematical formula is used to project the effort required based on estimates of product attributes, such as size, and process characteristics, such as experience of staff involved.

In both cases, you need to use your judgment to estimate either the effort directly, or estimate the project and product characteristics. In the taskup phase of a project, these estimates have a wide margin of error. Based on data collected from a large number of projects, Boehm, et al. (1991) discovered that start-up estimates vary significantly. If the initial estimate of effort required is a model of effort, they found that the range may be from 0.25 to 4 of the actual effort as measured when the system was delivered. During development planning, estimates become more and more accurate as the project progresses (Figure 23.39).

Experience-based techniques rely on the manager’s experience of past projects and the actual effort expended on these projects on activities that are related to software development. Typically, you identify the deliverables to be produced in a project and the different software components or systems that are to be developed. You document these in a spreadsheet, estimate them individually and compute the local effort required. It usually helps to get a group of people involved in the effort estimation and to ask each member of the group to explain their estimate. This often reveals facts that others have not considered and you then iterate towards an agreed group estimate.

The difficulty with experience-based techniques is that a new software project may not have much in common with previous projects. Software development changes very quickly and a project will often use unfamiliar techniques such as web services, CORBA-based development, or AIX. If you have not worked with these techniques, your previous experience may not help you to estimate the effort required.

Most algorithmic estimation models have an exponential component (B in the above equation) that is related to the size and complexity of the system. This reflects the fact that costs do not usually increase linearly with project size. As the size and complexity of the software increases, extra costs are incurred because of the communication overhead of larger teams, more complex configuration management, more difficult system integration, and so on. The more complex the system, the more these factors affect the cost. Therefore, the value of B usually increases with the size and complexity of the system.

All algorithmic models have similar problems:

1. It is often difficult to estimate S/M at an early stage in a project, when only the specification is available. Function-point and application-point estimates (earlier) are easier to produce than estimates of code size but are still often inaccurate.

2. The estimates of the factors contributing to B and M are subjective. Estimates vary from one person to another, depending on their background and experience of the type of system that is being developed.

Accurate code size estimation is difficult at an early stage in a project because the size of the final program depends on design decisions that may not have been made when the estimate is required. For example, in an application that requires high-performance data management may either implement its own data management system or use a commercial database system. In the initial cost estimation, you are unable to know if there is a commercial database system that performs well enough to meet the performance requirements. You therefore don’t know how much data management code will be included in the system.

The programming language used for system development also affects the number of lines of code to be developed. A language like Java might mean that more lines of code are necessary than if C (say) was used. However, this extra code allows more complete checking on validation codes are likely to be reduced. How should this be taken into account? Furthermore, it may be possible to reduce a significant amount of code from previous projects and the size estimate has to be adjusted to take this into account.

Algorithmic cost models are systematic ways to estimate the effort required to develop a system. However, these models are complex and difficult to use. There are many attributes and considerable scope for uncertainty in estimating their values. This complexity discourages potential users and hence the practical application of algorithmic cost modeling has been limited to a small number of companies.

Another barrier that discourages the use of algorithmic models is the need for calibration. Model users should calibrate their model and the attribute values using their own historical project data, as this reflects local practice and experience. However, very few organizations have collected enough data from past projects in a form that supports model calibration. Practical use of algorithmic models, therefore, has to start with the published values for the model parameters. It is practically impossible for a modeler to know how closely these relate to their own organization.

If you use an algorithmic cost estimation model, you should develop a range of esti-
Software productivity

Software productivity is an estimate of the average amount of development work that software engineers complete in a week or a month. It is therefore expressed as lines of code/month, function points/month, etc. However, whilst productivity can be easily measured where there is a tangible outcome (e.g., a desk processes in inches/day), software productivity is more difficult to define. Different people may implement the same functionality in different ways, using different numbers of lines of code, and the quality of the code is also important, but it is to some extent subjective. Productivity comparisons between software engineers are, therefore, unreliable and so are not very useful for project planning.

http://www.softwareengineering-uk.com/whick/Planning/productivity.html

Table 23.11 Application-point productivity

<table>
<thead>
<tr>
<th>Developer's experience and capability</th>
<th>Very low</th>
<th>Low</th>
<th>Normal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO maturity and capability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROD (LAP/month)</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

The application-composition model

The application-composition model was introduced into COCOMO II to support the estimation of effort required for prototyping projects and for projects where the software is developed by composing existing components. It is based on a weighted average of application points. Each application point is divided by a standard estimate. The application points are then added together to give the total effort required.

PM = (NAP × (1 - Reuse/100)) / PROD

Where PM is the effort estimate in person-months, NAP is the total number of application points in the delivered system, and Reuse is an estimate of the amount of reused code in the development. PROD is the application-point productivity, as shown in Figure 23.11. The model provides an approximate estimate as it does not take into account the additional effort involved in reuse.

The early design model

The model may be used during the early stages of a project, before a detailed architectural design for the system is available. Early design estimates are most useful for option exploration where you need to compare different ways of implementing the

...
the additional effort required to reuse code. Simplistically, AAM is the sum of three components:

1. An adaptation component (referred to as AAF) that represents the costs of making changes to the reused code. The adaptation component includes sub-components that take into account design, code, and integration changes.

2. An understanding component (referred to as SU) that represents the costs of understanding the code to be reused and the familiarity of the engineers with the code. SU ranges from 0 for complete unstructured code to 10 for well-written, object-oriented code.

3. An assessment factor (referred to as AA) that represents the costs of reuse decision making. That is, some analysis is always required to decide whether or not code can be reused, and this is included in the cost as AA. AA varies from 0 to 8 depending on the amount of analysis effort required.

If some code adaptation can be done automatically, this reduces the effort required. You therefore adjust the estimate by the percentage of automatically adapted code (AA) and using this to adjust ASLOC. Therefore, the final formula is:

\[ \text{ESLOC} = \text{ASLOC} \times (1 - \text{AA/100}) \times \text{AAM} \]

Once ESLOC has been calculated, you then apply the standard estimation formula to calculate the total effort required, where the Size parameter = ESLOC. You then add this to the effort to integrate automatically generated code that you have already computed, thus computing the total effort required.

The post-architecture level

The post-architecture model is the most detailed of the Cocomo II models.

![Table of estimation factors](Image)

### 23.5.3 Project duration and staffing

As well as estimating the overall costs of a project and the effort that is required to develop a software system, project managers and estimate how long the software will take to develop, and when staff will be needed to work on the project. Formally, organizations are demanding shorter development schedules so that their products can be brought to market before their competitor's.

The Cocomo II model includes a formula to estimate the calendar time required to complete a project:

\[ T = \frac{S}{F} \]

Where:

- \( T \) = The calendar time required to complete the project
- \( S \) = The scale factor calculated by the Cocomo II model
- \( F \) = The efficiency factor (the scale factor calculated by the Cocomo II model)

### 23.5.4 Estimation techniques

- **PM (Project Management)**: This is the primary tool used in project management. PM focuses on the efficient use of resources to achieve the project's objectives.
- **BSM (Baseline System Model)**: This is a document that describes the high-level architecture of the system.
- **COCOMO II models**: These are predictive models that estimate the effort and cost of software development projects.

### 23.5.5 Project duration and staffing

The Cocomo II model includes a formula to estimate the calendar time required to complete a project:

\[ T = \frac{S}{F} \]

Where:

- \( T \) = The calendar time required to complete the project
- \( S \) = The scale factor calculated by the Cocomo II model
- \( F \) = The efficiency factor (the scale factor calculated by the Cocomo II model)
However, the nominal project schedule predicted by the COCOMO model and the schedule required by the project plan are not necessarily the same thing. There may be a requirement to deliver the software earlier or more rapidly than the date suggested by the nominal schedule. If the schedule is to be controlled, this increases the effort required for the project. This is taken into account by the SCED multiplier in the effort estimation component.

Assume that a project estimated TRED as 13 months, as suggested above, but the actual schedule required was 11 months. This represents a schedule compression of approximately 23%. Using the values for the SCED multiplier as derived by Boehm’s team, the effort multiplier for such a schedule compression is 1.43. Therefore, the actual effort that will be required (if this accelerated schedule is to be met) is almost 50% more than the effort required to deliver the software according to the nominal schedule.

There is a complex relationship between the number of people working on a project, the effort that will be devoted to the project, and the project delivery schedule. If four people can complete a project in 13 months (i.e., 52 person-months of effort), then you might think that by adding one more person, you can complete the work in 11 months (55 person-months of effort). However, the COCOMO model suggests that you will, in fact, need six people to finish the work in 11 months (66 person-months of effort).

The reason for this is that adding more people actually reduces the productivity of existing team members and so the actual increment of effort added is less than one person. As the project team increases in size, team members spend more time communicating and forming interfaces between parts of the system developed by other people. Doubling the number of staff (for example) therefore does not mean that the duration of the project will be halved. If the development team is large, it is sometimes the case that adding more people to a project increases rather than reduces the development schedule. Myers (1989) discusses the problems of schedule acceleration. He suggests that projects are likely to run into significant problems if they try to develop software without allowing sufficient calendar time to complete the work.

You cannot simply estimate the number of people required for a project team by dividing the total effort by the required project schedule. Usually, a small number of people are needed at the start of a project to carry out the initial design. The team then builds up to a peak during the development and testing of the system, and then declines in size as the system is prepared for deployment. A very rapid buildup of project staff has been shown to correlate with project schedule slippage. Project managers should therefore avoid adding too many staff to a project early in its lifetime.

This effort buildup can be modeled by what is called a Rayleigh curve (Lumelsky, 1987). Petten’s estimation model (1978), which incorporates a model of project staffing, is based around these Rayleigh curves. This model also includes development time as a key factor. As development time is reduced, the effort required to develop the system grows exponentially.

EXERCISES

23.1. Under what circumstances might a company justifyably charge a much higher price for a software system than the software cost estimate plus a reasonable profit margin?

23.2. Explain why the process of project planning is iterative and why a plan must be continually reviewed during a software project.

23.3. Briefly explain the purpose and reach of the sections in a software project plan.

23.4. Cost estimates are inherently risky, irrespective of the estimation technique used. Suggest four ways in which the risk in a cost estimate can be reduced.

23.5. Figure 23.14 shows the number of tasks, their durations, and their dependencies. Draw a bar chart showing the project schedule.

23.6. Figure 23.14 shows the task durations for software project activities. Assume that a serious, unexpected setback occurs and instead of taking 10 days, task T5 takes 40 days. Draw a new bar chart showing how the project might be reengineered.

23.7. The XP planning game is based around the notion of planning to implement the stories that represent the system requirements. Explain the potential problems with this approach when software has high performance or dependability requirements.

23.8. A software manager is in charge of the development of a safety-critical software system, which is designed to control a radiotherapy machine to treat patients suffering from cancer. This system is embedded in the machine and must run on a special-purpose processor with a fixed amount of memory (156 MB). The machine communicates with a patient database system to obtain the details of the patient and, after treatment, automatically records the radiation dose delivered and other treatment data in the database.

The COCOMO method is used to estimate the effort required to develop this system and an estimate of 26 person-months is computed. All cost driver multipliers were set to 1 when making this estimate. Explain why this estimate should be adjusted to take project, personnel, product, and organizational factors into account. Suggest four factors that might have significant effects on the initial COCOMO estimate and propose possible values for these factors. Justify why you have included each factor.

23.9. Some very large software projects involve writing millions of lines of code. Explain why the effort estimation models such as COCOMO might not work well when applied to very large systems.

23.10. Is it ethical for a company to quote a low price for a software contract knowing that the requirements are ambiguous and that they can charge a high price for subsequent changes requested by the customer?

REFERENCES


Quality management

Objectives
The objectives of this chapter are to introduce software quality management and software measurement. When you have read the chapter, you will:

- have been introduced to the quality management process and know why quality planning is important;
- understand that software quality is affected by the software development process used;
- be aware of the importance of standards in the quality management process and know how standards are used in quality assurance;
- understand how reviews and inspections are used as a mechanism for software quality assurance;
- understand how measurement may be helpful in assessing some software quality attributes and the current limitations of software measurement.

Contents
24.1 Software quality
24.2 Software standards
24.3 Reviews and inspections
24.4 Software measurement and metrics