23 Project planning pg 618-633

Objectives

The objective of this chapter **is to introduce** project **planning, scheduling**,

When you have read the chapter, you will:

- ♉ understand the fundamentals of software costing and reasons why the price of the software may not be directly related to its development cost;
- ♉ know what sections should be included in a project plan that is created within a plan-driven development process;
- ♉ understand what is involved in project scheduling and the use of bar charts to present a project schedule;
- ♉ have been introduced to the ‘planning game’, which is used to support project planning in extreme programming;

.

Contents

23.1 Software pricing
23.2 Plan-driven development
23.3 Project scheduling
23.4 Agile planning

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 these to project team members, anticipate problems that might arise, and prepare tentative solutions to those 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 who will work 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 and this usually means that the work breakdown has to be altered and the schedule extended. For traditional development projects, this means that the plan created during the startup phase has to be modified.** However, **when an agile approach is used**, **plans are shorter term and continually** change as the software evolves.

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.**

Overhead costs
When you estimate the costs of effort on a software project, **you don't simply multiply the salaries** of the people involved by the time spent on the project. **You have to take into account all of the organizational overheads (office space, administration, etc.)** that must be covered by the income from a project. You calculate the costs by computing these overheads and adding a proportion to the costs of each engineer working on a project.
http://www.SoftwareEngineering-9.com/Web/Planning/overheadcosts.html

Many factors influence the pricing of a software project—it is not simply cost + profit.

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

1. effort costs (the costs of paying software engineers and managers);
2. hardware and software costs, including maintenance;
3. 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 acomplete requirements specification, especially if you are using an agile approach to development.

Your aim 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 good
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 slippage to be reported, the project manager could then
immediately assign an 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

1. **schedule,**
2. **cost estimate, and**
3. **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, as regardless
of 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.

23.1 Software pricing

In principle**, the price of a software product to a customer**

1. **is simply the cost of development**
2. **plus profit for the developer.**

In practice, however, **the relationship between the project cost and the price quoted to the customer is not usually so simple**.

When calculating a price, **you should**

1. take broader organizational,
2. economic,
3. political, and
4. business considerations

into account.
You **need to think about**

1. **organizational concerns,**
2. **the risks associated with the project,**
3. **and the type of contract that will be used.**

These may cause the price to be adjusted upwards or downwards.
Because of the organizational considerations involved, deciding on a project price

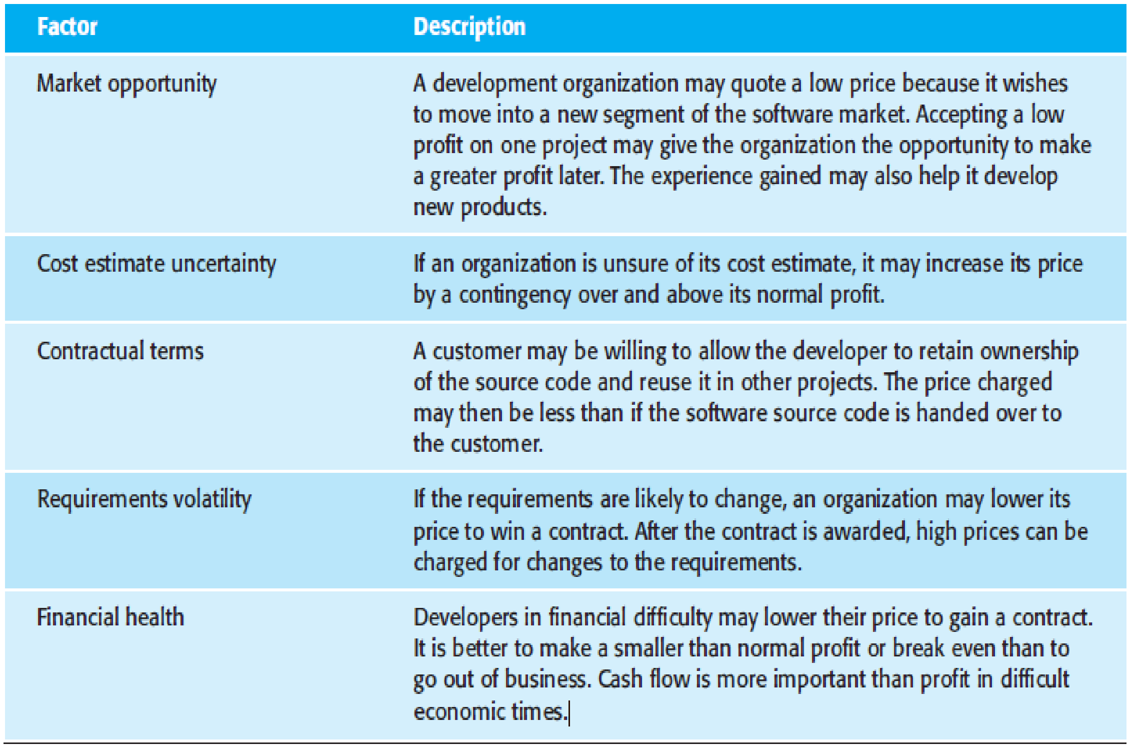
1. **should be a group activity involving marketing and sales staff,**
2. **senior management,**
3. **and project managers.**

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 competitive position, PharmaSoft decides to bid a price to the customer of $0.8 million.



Factor	Description
Market opportunity	A development organization may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Cost estimate uncertainty	If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business. Cash flow is more important than profit in difficult economic times.

Figure 23.1 Factors affecting software pricing

Market opportunity

A development organization may quote a low price because it wishes to move into a new segment of the software market. **Accepting a low profit on one project may give the organization the opportunity to make a greater profit later.** The experience gained may also help it develop new products.

Cost estimate uncertainty

If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.

Contractual terms

A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.

Requirements volatility

If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.

Financial health

Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business. Cash flow is more important than profit in difficult economic times.

This means that, although it loses 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 a project is only loosely related to the price quoted to 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 expected price. This may seem unethical and unbusinesslike, 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 not the project requirements but the cost**, The **requirements may be changed so that the cost is not exceeded**.

The additional requirements **may be funded from a future budget**, so that the a company's **budgeting is not disrupted by a high initial software cost**.

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 revised 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

1. **allows organizational issues** (availability of staff, other projects, etc.) **to be closely taken into account**,

2. **and that potential problems and dependencies are discovered before the project starts, rather than once the project is under way.**

It seems that **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 skills of the people who are available.

1. **At one extreme**, **large security 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.**

2. **At the other extreme**, small to 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

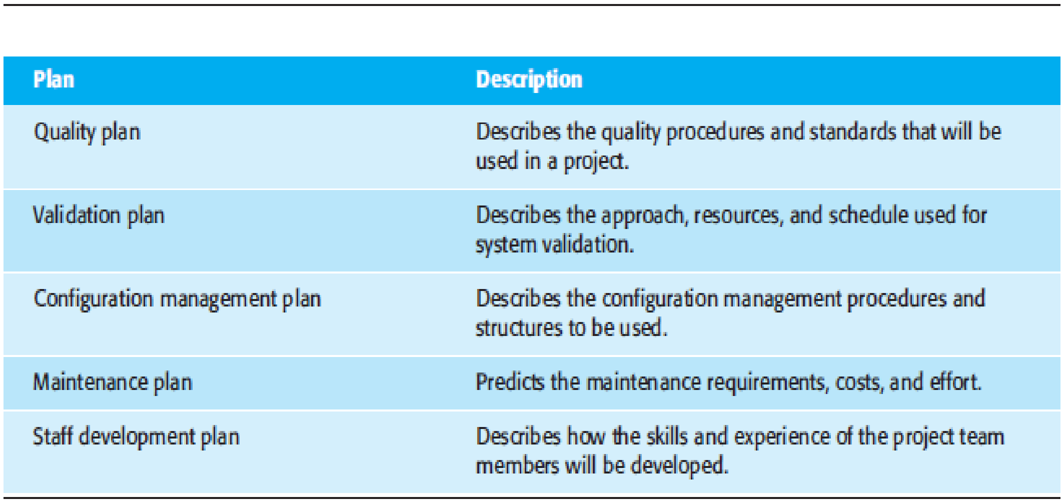
In a plan-driven development project, a project plan sets out

- **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 organization, plans normally **include the following sections:**

1. *Introduction* This briefly describes the objectives of the project and sets out the constraints (e.g., budget, time, etc.) that affect the management of the project.



Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project.
Validation plan	Describes the approach, resources, and schedule used for system validation.
Configuration management plan	Describes the configuration management procedures and structures to be used.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
Staff development plan	Describes how the skills and experience of the project team members will be developed.

Figure 23.2 Project plan supplements

Quality plan Describes the quality procedures and standards that will be used in a project.
Validation plan Describes the approach, resources, and schedule used for system validation.
Configuration management plan Describes the configuration management procedures and structures to be used.
Maintenance plan Predicts the maintenance requirements, costs, and effort.
Staff development plan Describes how the skills and experience of the project team members will be developed.

2. Project organization This describes the way in which the development team is organized, the people involved, and their roles in the team.

3. Risk analysis This describes possible project risks, the likelihood of these risks arising, and the risk reduction strategies that are proposed.

4. Hardware and software resource requirements This specifies the hardware and support software required to carry out the development. If hardware has to be bought, estimates of the prices and the delivery schedule may be included.

5. Work breakdown This sets out the breakdown of the project into activities and identifies the milestones and deliverables associated with each activity. Milestones are key stages in the project where progress can be assessed; deliverables are work products that are delivered to the customer.

6. Project schedule This shows the dependencies between activities, the estimated time required to reach each milestone, and the allocation of people to activities. The ways in which the schedule may be presented are discussed in the next section of the chapter.

7. Monitoring and reporting mechanisms This defines the management reports that should be produced, when these should be produced, and the project monitoring mechanisms to be used.

As well as the principal project plan, which should focus on the risks to the projects and the project schedule, you may develop a number of supplementary plans to support other process activities such as testing and configuration management.

Examples of possible supplementary plans are shown in Figure 23.2.

23.2.2 The planning process

Project planning is an iterative process that starts when you create an initial project plan during the project startup phase. Figure 23.3 is a UML activity diagram that

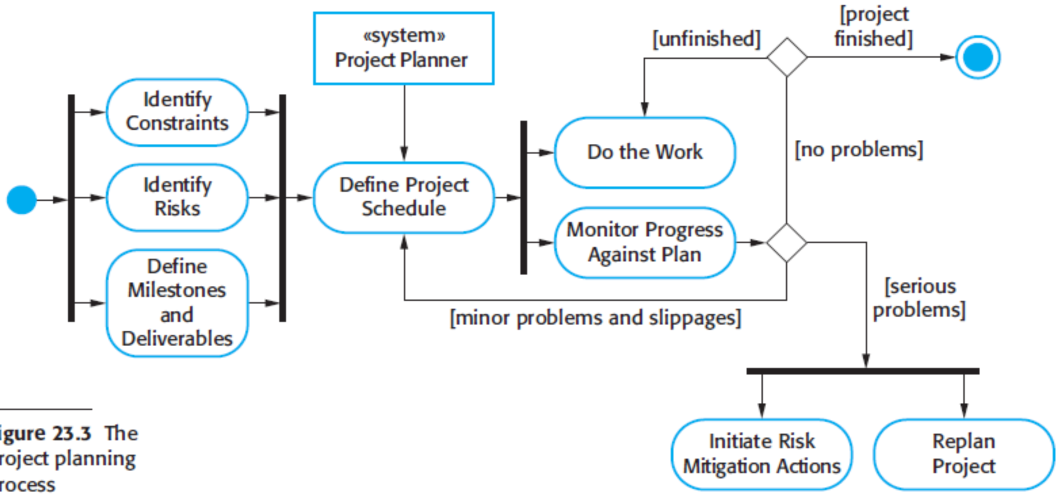


Figure 23.3 The project planning process

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 revise the plan to reflect requirements, schedule, and risk changes. Changing business goals also leads to changes in project plans. As business goals change, this could affect all projects, which may then have to be replanned.
At the beginning of a planning process, you should assess 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 (e.g., a requirements document 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 description 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 constraints 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 project failure. In conjunction with these actions, you also have to replan the project.

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 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 failings but, more often, is a consequence of external changes that affect the project. The development time for a large software project is often several years. During that time, the business objectives and priorities 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 travel 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 has to be an overall schedule that identifies when the major phases of the project will be completed. **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**.
Finer subdivision means that a disproportionate amount of time must be spent on replanning and updating the project plan.

Activity charts
An activity chart is a project schedule representation that shows which tasks can be carried out in parallel and those that must be executed in sequence, due to their dependencies on earlier activities. If a task is dependent on several other tasks then all of these must finish before it can start. The 'critical path' through the activity chart is the longest sequence of dependent tasks. This defines the project duration.
http://www.SoftwareEngineering-9.com/Web/Planning/activities.html

The maximum amount of time for any task **should be around 8 to 10 weeks**. If it takes longer than this, the task should be subdivided for project planning and scheduling.

Some of these tasks are carried out in parallel, with different people working on different components of the system. You have to coordinate these parallel tasks and organize the work so that the workforce is used optimally and you don't introduce unnecessary dependencies between the tasks. It is important to avoid a situation where the whole project is delayed because a critical task is unfinished.
If a project is technically advanced, initial estimates will almost certainly be optimistic even when you try to consider all eventualities. In this respect, software scheduling is no different from scheduling any other type of large advanced project. New aircraft, bridges, and even new models of cars are frequently late because of unanticipated problems. Schedules, therefore, must be continually updated as better progress information becomes available. If the project being scheduled is similar to a previous project, previous estimates may be reused. However, projects may use different design methods and implementation languages, so experience from previous projects may not be applicable in the planning of a new project.
As I have already suggested, when you are estimating schedules, you must take into account the possibility that things will go wrong. People working on a project may fall ill or leave, hardware may fail, and essential support software or hardware may be delivered late. If the project is new and technically advanced, parts of it may turn out to be more difficult and take longer than originally anticipated.
A good rule of thumb is to estimate as if nothing will go wrong, then increase your estimate to cover anticipated problems. A further contingency factor to cover unanticipated problems may also be added to the estimate. This extra contingency factor depends on the type of project, the process parameters (deadline, standards, etc.), and the quality and experience of the software engineers working on the project. Contingency estimates may add 30% to 50% to the effort and time required for the project.

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 style of representation makes it difficult to see the relationships and dependencies

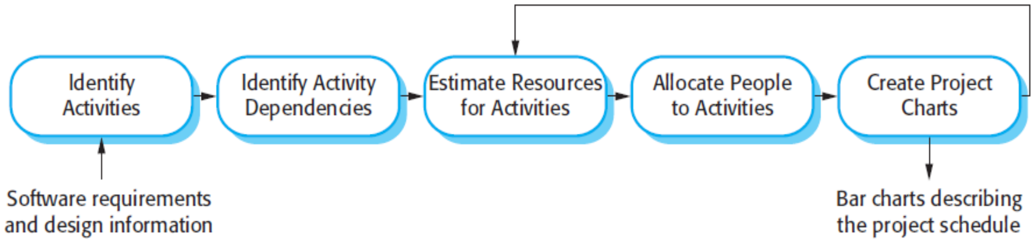

Identify Activities
Identify Activity Dependencies
Estimate Resources for Activities
Allocate People to Activities
Create Project Charts
Software requirements and design information
Bar charts describing the project schedule

Figure 23.4 The project scheduling process

between the different activities. For this reason, alternative graphical representations of project schedules have been developed that are often easier to read and understand.

There are two types of representation that are commonly used:

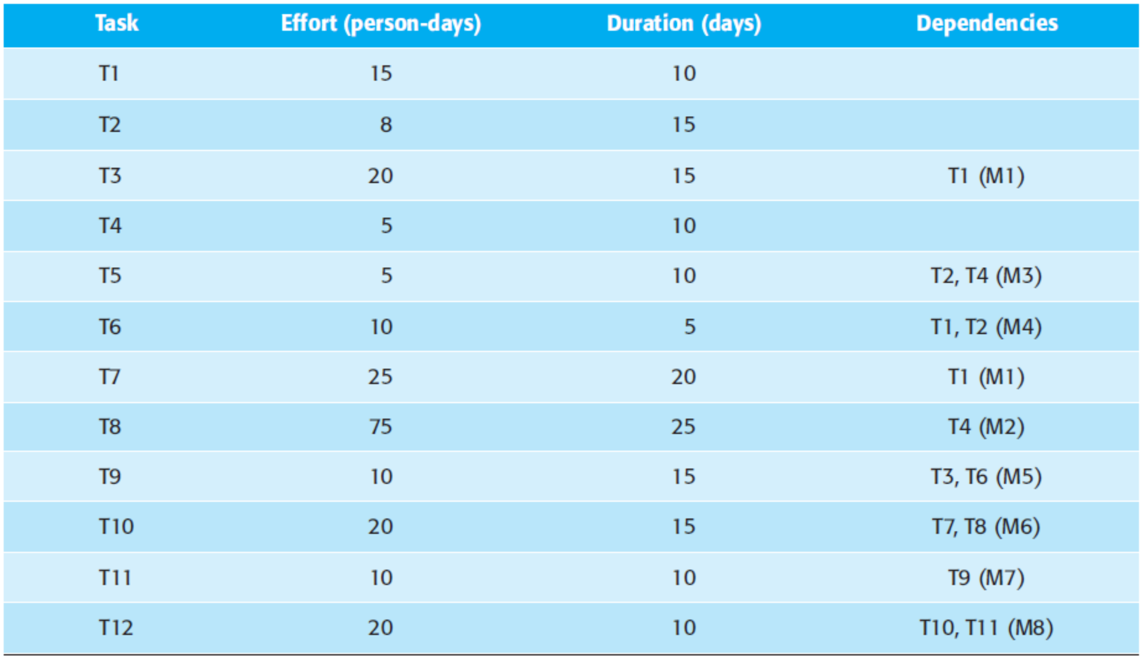
1. **Bar charts, which are calendar-based**, show who is responsible for each activity, the expected elapsed time, and when the activity is scheduled to begin and end. Bar charts are sometimes called 'Gantt charts', after their inventor, Henry Gantt.

2. Activity networks, which are network diagrams, show the dependencies between the different activities making up a project.

Normally, a project planning tool is used to manage project schedule information. These tools usually expect you to input project information into a table and will then create a database of project information. Bar charts and activity charts can then be generated automatically from this database.

Project activities are the basic planning element. Each activity has:
1**. A duration in calendar days or months**.
2. **An effort estimate, which reflects the number of person-days or person-months to complete the work**.
3. **A deadline by which the activity should be completed**.
4. **A defined endpoint.** This represents the tangible result of completing the activity. This could be a document, the holding of a review meeting, the successful execution of all tests, etc.

When planning a project, you should also define milestones; that is, each stage in the project where a progress assessment can be made. Each milestone should be documented by a short report that summarizes the progress made and the work done. Milestones may be associated with a single task or with groups of related activities. For example, in Figure 23.5, milestone M1 is associated with task T1 and milestone M3 is associated with a pair of tasks, T2 and T4.
A special kind of milestone is the production of a project deliverable. A deliverable is a work product that is delivered to the customer. It is the outcome of a significant project phase such as specification or design. Usually, the deliverables that are



Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Figure 23.5 Tasks, durations, and dependencies

required are specified in the project contract and the customer's view of the project's progress depends on these deliverables.

To illustrate how bar charts are used, I have created a hypothetical set of tasks as shown in Figure 23.5. This table shows tasks, estimated effort, duration, and task interdependencies.

From Figure 23.5, you can see that task T3 is dependent on task T1.

Task T1 must, therefore, be completed before T3 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, 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 23.6 takes the information in Figure 23.5 and presents the project schedule 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 independent 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 is, of course, 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, which shows when staff are working on the project (Figure 23.7). People may be

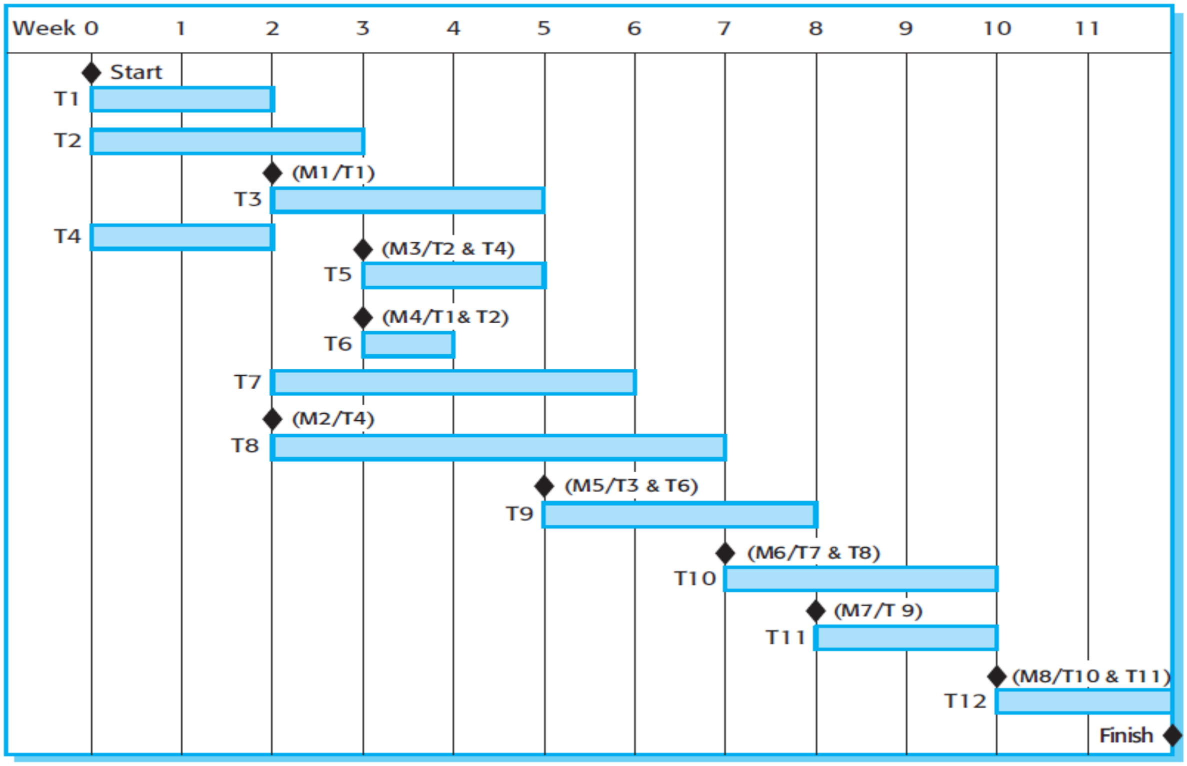


Figure 23.6
Activity bar chart

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 may be assigned to other work (W). To complete this 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 as they complete W.

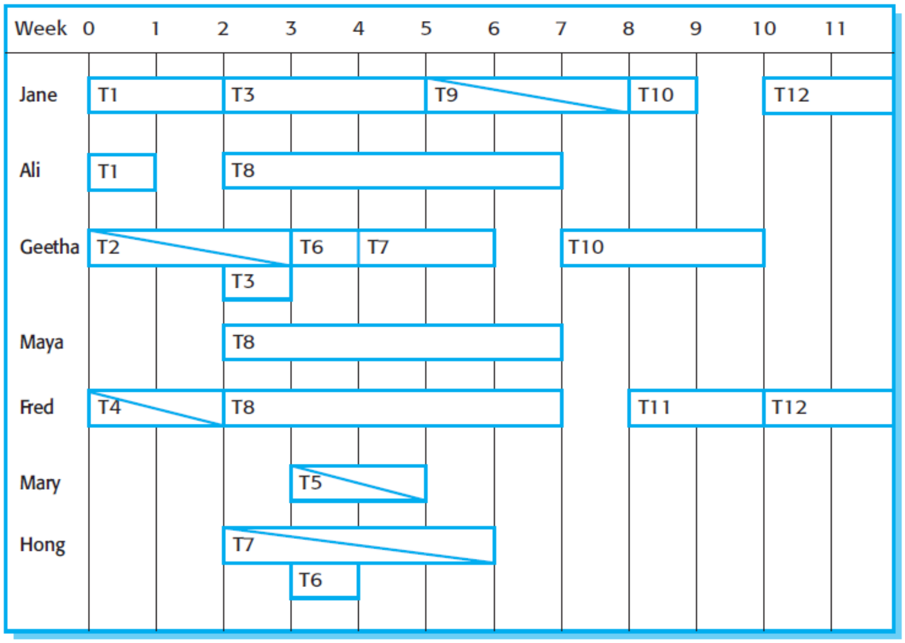


Figure 23.7 Staff allocation chart

23.4 Agile planning

Agile methods of software development are iterative approaches where the software is developed and delivered to customers in increments.

Unlike plan-driven approaches, the functionality of these increments is not planned in advance but is decided during the development. The decision on what to include in an increment depends on progress and on the customer's priorities. The argument for this approach is that the customer's priorities and requirements change so it makes sense to have a flexible plan that can accommodate these changes. Cohn's book {Cohn, 2005 ##1735} is a comprehensive discussion of planning issues in agile projects. The most commonly used agile approaches such as Scrum (Schwaber, 2004) and extreme programming (Beck, 2000) have a two-stage approach to planning, corresponding to the startup phase in plan-driven development and development planning:

1. Release planning, which looks ahead for several months and decides on the features that should be included in a release of a system.
2. Iteration planning, which has a shorter-term outlook, and focuses on planning the next increment of a system. This is typically 2 to 4 weeks of work for the team.

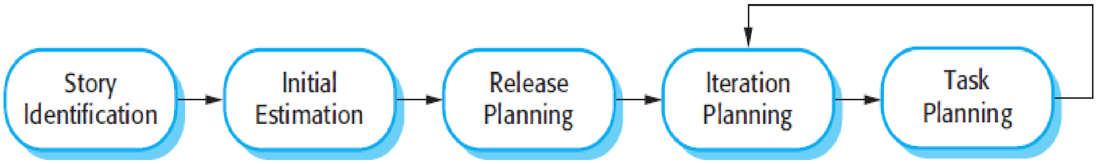


Figure 23.8
Planning in XP

I have already discussed the Scrum approach to planning in Chapter 3, so I concentrate

Scrum
An agile method of development, which is based on sprints—short development, cycles. Scrum may be used as a basis for agile project management alongside other agile methods such as XP.
Scrum approach and, 56, 57, 59, 72–74, 78,
631, 632

3 Extreme programming

Extreme programming (XP) is perhaps the best known and most widely used of the agile methods. The name was coined by Beck (2000) because the approach was developed by pushing recognized good practice, such as iterative development, to 'extreme' levels. For example, in XP, several new versions of a system may be developed by different programmers, integrated and tested in a day. Pg 64

here on planning in extreme programming (XP). This is called 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 user stories that reflect 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 covers 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 they 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 then allocates notional effort points to the stories. A complex story may have 8 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 reflect the features to be implemented in a release of a system and the order in which the stories should be implemented. The customer has to be involved in this process. A release date is then chosen and the stories are examined to see if the effort estimate is consistent with that date. If not, stories are added or removed from the list.
Iteration planning is the first stage into the iteration development process. Stories to be implemented for that iteration are chosen, with the number of stories reflecting the time to deliver an iteration (usually 2 or 3 weeks) and the team's velocity. When the iteration delivery date is reached, that iteration is complete, even if all of the stories have not been implemented. The team considers the stories that have been implemented and adds up their effort points. The velocity can then be recalculated

and this is used in planning the next release of the system.
At the start of each iteration, there is a more detailed planning stage where the developers break stories down into development tasks. A development task should

take 4–16 hours. All of the tasks that must be completed to implement all of the stories in that iteration are listed. The individual developers then sign up for the specific tasks that they will implement. Each developer knows their individual velocity so should not sign up for more tasks than they can implement in the time.
There are two important benefits from this approach to task allocation:
1. The whole team gets an overview of the tasks to be completed in an iteration. They therefore have an understanding of what other team members are doing and who to talk to if task dependencies are identified.
2. Individual developers choose the tasks to implement; they are not simply allocated tasks by a project manager. They therefore have a sense of ownership in these tasks and this is likely to motivate them to complete the task.
Halfway though an iteration, progress is reviewed. At this stage, half of the story effort points should have been completed. So, if an iteration involves 24 story points and 36 tasks, 12 story points and 18 tasks should have been completed. If this is not the case, then the customer has to be consulted and some stories removed from the iteration.
This approach to planning has the advantage that the software is always released as planned and there is no schedule slippage. If the work cannot be completed in the time allowed, the XP philosophy is to reduce the scope of the work rather than extend the schedule. However, in some cases, the increment may not be enough to be useful. Reducing the scope may create extra work for customers if they have to use an incomplete system or change their work practices between one release of the system and another.
A major difficulty in agile planning is that it is reliant on customer involvement and availability. In practice, this can be difficult to arrange, as the customer representative must sometimes give priority to other work. Customers may be more familiar with traditional project plans and may find it difficult to engage in an agile planning project.
Agile planning works well with small, stable development teams that can get together and discuss the stories to be implemented. However, where teams are large and/or geographically distributed, or when team membership changes frequently, it is practically impossible for everyone to be involved in the collaborative planning that is essential for agile project management. Consequently, large projects are usually planned using traditional approaches to project management.

Pentru prelegerea urmatoare, de la pagina 633