staff might reveal that the team is having problems with a software fault in the communications systems. The project manager can then immediately assign a communications expert to the problem to help find and solve the problem.

The project plan always evolves during the development process because of requirements changes, technology issues, and development problems. 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 because 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 you only need to include essential 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. Some aspects of agile planning have already been covered in Chapter 3, and I discuss other approaches in Section 23.4.

23.1 Software pricing

In principle, the price of a software system developed for 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 so simple. When calculating a price, you take broader organizational, economic, political, and business considerations into account (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 issues may cause the price to be adjusted upward or downward.

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. 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.

Factor	Description
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 might then be reduced to reflect the value of the source code to the developer.
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.
Financial health	Companies with financial problems 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.
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.
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.

Figure 23.1 Factors affecting software pricing

This is an example of an approach to software pricing called "pricing to win." 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 project costs remain within budget.

For example, say a company (OilSoft) 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 OilSoft estimates that a price of $900,000 is likely to be competitive and within the oil company's budget. After being granted the contract, OilSoft then negotiates the detailed requirements of the system so that basic functionality is delivered. It then estimates the additional costs for other requirements.

This approach has advantages for both the software developer and the customer. The requirements are negotiated to avoid requirements that are difficult to implement and potentially very expensive. Flexible requirements make it easier to reuse software. The oil company has awarded the contract to a known company that it can trust. Furthermore, it may be possible to spread the cost of

the project over several versions of the system. This may reduce the costs of system deployment and allow the client to budget for the project cost over several financial years.

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. Agile development involves a different planning process, discussed in Section 23.4, where decisions are delayed.

The problem with plan-driven development is that early decisions have to be revised because of changes to the environments in which the software is developed and used. Delaying planning decisions avoids unnecessary rework. However, the arguments in favor of a plan-driven approach are that early planning allows organizational issues (availability of staff, other projects, etc.) to be taken into account. Potential problems and dependencies are discovered before the project starts, rather than once the project is underway.

In my view, the best approach to project planning involves a sensible mixture of plan-based and agile development. The balance depends on the type of project and skills of the people who are available. 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 systems should be mostly plan-driven. 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 the approach that is taken to risk management as well as risks to the project and the software under development. The details of project plans vary depending on the type of project and organization but plans normally include the following sections:

1. *Introduction* Briefly describes the objectives of the project and sets out the constraints (e.g., budget, time) that affect the management of the project.
2. *Project organization* Describes the way in which the development team is organized, the people involved, and their roles in the team.

Plan	Description
Configuration management plan	Describes the configuration management procedures and structures to be used.
Deployment plan	Describes how the software and associated hardware (if required) will be deployed in the customer's environment. This should include a plan for migrating data from existing systems.
Maintenance plan	Predicts the maintenance requirements, costs, and effort.
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.

Figure 23.2 Project plan supplements

3. *Risk analysis* Describes possible project risks, the likelihood of these risks arising, and the risk reduction strategies (discussed in Chapter 22) that are proposed.
4. *Hardware and software resource requirements* Specifies the hardware and support software required to carry out the development. If hardware has to be purchased, estimates of the prices and the delivery schedule may be included.
5. *Work breakdown* Sets out the breakdown of the project into activities and identifies the inputs to and the outputs from each project activity.
6. *Project schedule* 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* Defines the management reports that should be produced, when these should be produced, and the project monitoring mechanisms to be used.

The main project plan should always include a project risk assessment and a schedule for the project. In addition, you may develop a number of supplementary plans for activities such as testing and configuration management. Figure 23.2 shows some supplementary plans that may be developed. These are all usually needed in large projects developing large, complex systems.

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 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

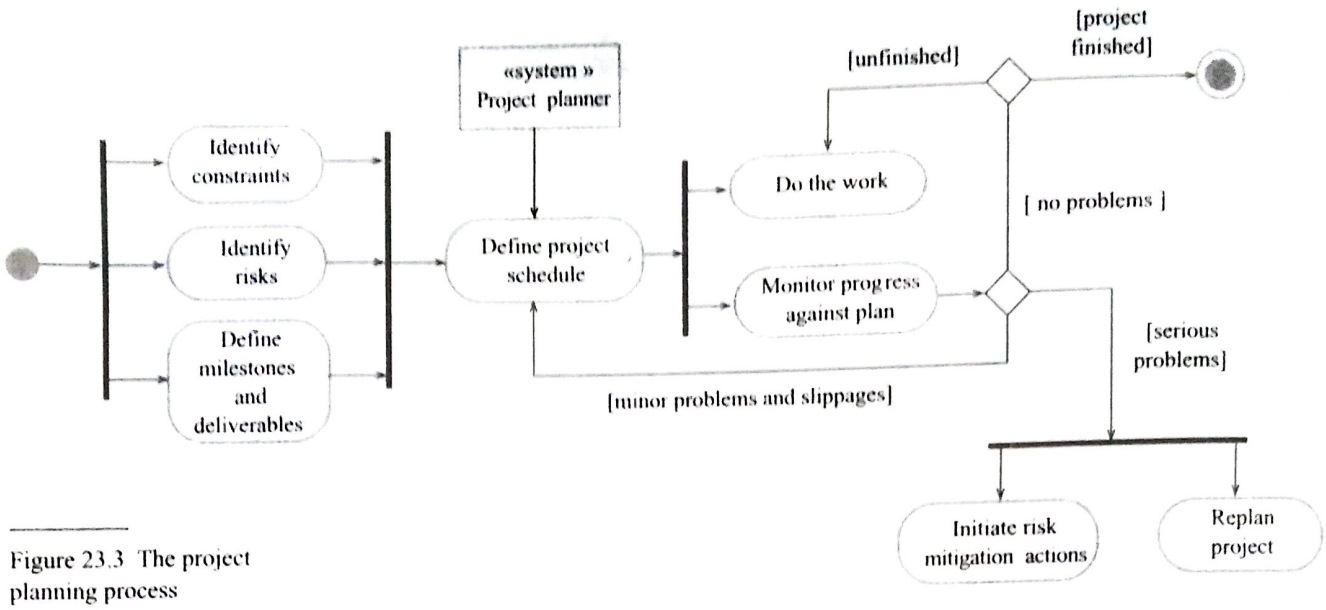

Figure 23.3 The project planning process

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 re-planned.

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 assessment, 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, for example, a requirements document for the system.

The process then enters a loop that terminates when the project is complete. You draw up an estimated schedule for the project, and the activities defined in the schedule are initiated or are approved 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.

You should make realistic rather than optimistic assumptions when you are defining a project plan. Problems of some description always arise during a project, and these lead to project delays. Your initial assumptions and scheduling should therefore be pessimistic and take unexpected problems into account. You should include contingency in your plan so that if things go wrong, then your delivery schedule is not seriously disrupted.

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 re-plan 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 to 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. Reviews should also check that the customer's goals are unchanged and that the project remains aligned with these goals.

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 and the effort required, and you suggest who will work on the tasks that have been identified. You also have to estimate the hardware and software resources that are needed to complete each task. For example, if you are developing an embedded system, you have to estimate the time that you need on specialized hardware and the costs of running a system simulator. In terms of the planning stages that I introduced 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 less detail is included 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 re-planning and updating the project plan. The maximum amount of time for any task should be 6 to 8 weeks. If a task will take longer than this, it should be split into subtasks 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

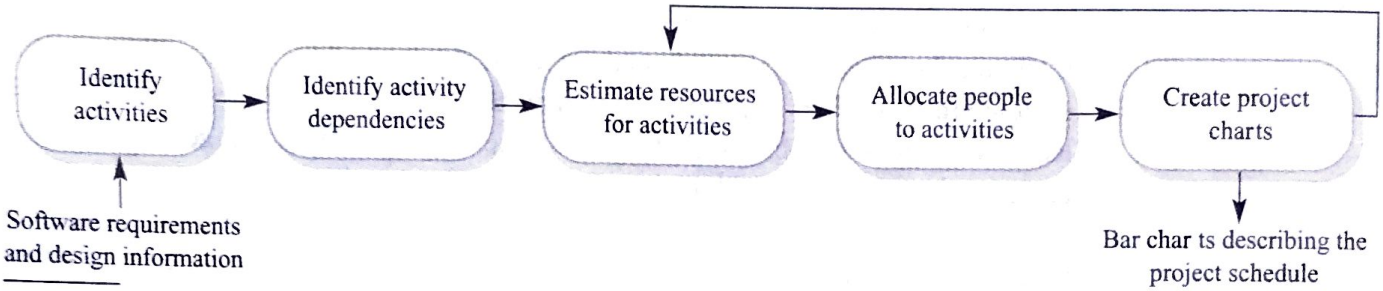

Figure 23.4 The project scheduling process

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.

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 and 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 presentation

Project schedules may simply be documented in a table or spreadsheet showing the tasks, estimated effort, duration, and task dependencies (Figure 23.5). However, this style of presentation makes it difficult to see the relationships and dependencies between the different activities. For this reason, alternative graphical visualizations of project schedules have been developed that are often easier to read and understand. Two types of visualization are commonly used:

1. Calendar-based bar charts show who is responsible for each activity, the expected elapsed time, and when the activity is scheduled to begin and end. Bar charts are also called Gantt charts, after their inventor, Henry Gantt.

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

2. Activity networks show the dependencies between the different activities making up a project. These networks are described in an associated web section.

Project activities are the basic planning element. Each activity has:

- a duration in calendar days or months;
- an effort estimate, which shows the number of person-days or person-months to complete the work;
- a deadline by which the activity should be complete; and
- a defined endpoint, which might be a document, the holding of a review meeting, the successful execution of all tests, or the like.

When planning a project, you may decide to define project milestones. A milestone is a logical end to a stage of the project where the progress of the work can be reviewed. Each milestone should be documented by a brief report (often simply an email) that summarizes the work done and whether or not the work has been completed as planned. 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 marks the end of that activity. Milestone M3 is associated with a pair of tasks T2 and T4; there is no individual milestone at the end of these tasks.

Activity charts

An activity chart is a project schedule representation that presents the project plan as a directed graph. It 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 tasks must be completed before it can start. The "critical path" through the activity chart is the longest sequence of dependent tasks. This defines the project duration.

http://software-engineering-book.com/web/planning-activities/

Some activities create project deliverables—outputs that are delivered to the software customer. Usually, the deliverables that are required are specified in the project contract, and the customer's view of the project's progress depends on these deliverables. Milestones and deliverables are not the same thing. Milestones are short reports that are used for progress reporting, whereas deliverables are more substantial project outputs such as a requirements document or the initial implementation of a system.

Figure 23.5 shows a hypothetical set of tasks, their estimated effort and duration, and task dependencies. From this table, you can see that task T3 is dependent on task T1. This means that task T1 has to be completed before T3 starts. For example, T1 might be the selection of a system for reuse and T3, the configuration of the selected system. You can't start system configuration until you have chosen and installed the application system to be modified.

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, 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 as 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 may be carried out in parallel. For example, 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. They have to be assigned to project activities. The resource allocation can be analyzed by project management tools, and a bar chart can be generated showing when staff are working on the project (Figure 23.7). People may be 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, or attending training courses. 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

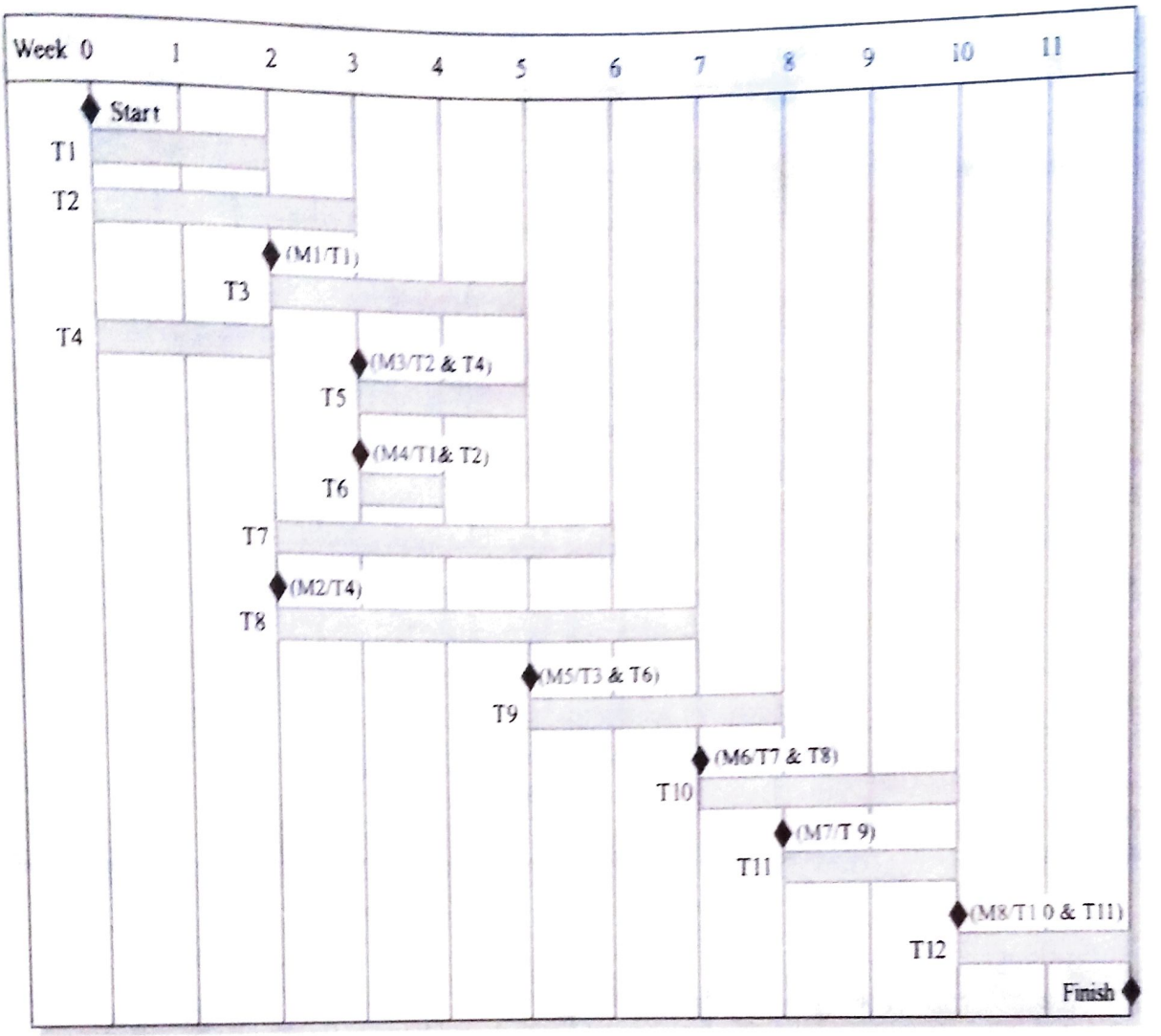

Figure 23.6 Activity bar chart

only a single task (T5) in the project. The use of specialists is unavoidable when complex systems are being developed, but it can lead to scheduling problems. If one project is delayed while a specialist is working on it, this may affect other projects where the specialist is also required. These projects may be delayed because the specialist is not available.

If a task is delayed, later tasks that are dependent on it may be affected. 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 (W). To complete this work may take longer than the delay, but, once assigned, they cannot simply be reassigned back to the original task. This may then lead to further delays in T as they complete W.

Normally, you should use a project planning tool, such as the Basecamp or Microsoft project, to create, update, and analyze project schedule information. Project management tools usually expect you to input project information into a table, and they create a database of project information. Bar charts and activity charts can then be generated automatically from this database.

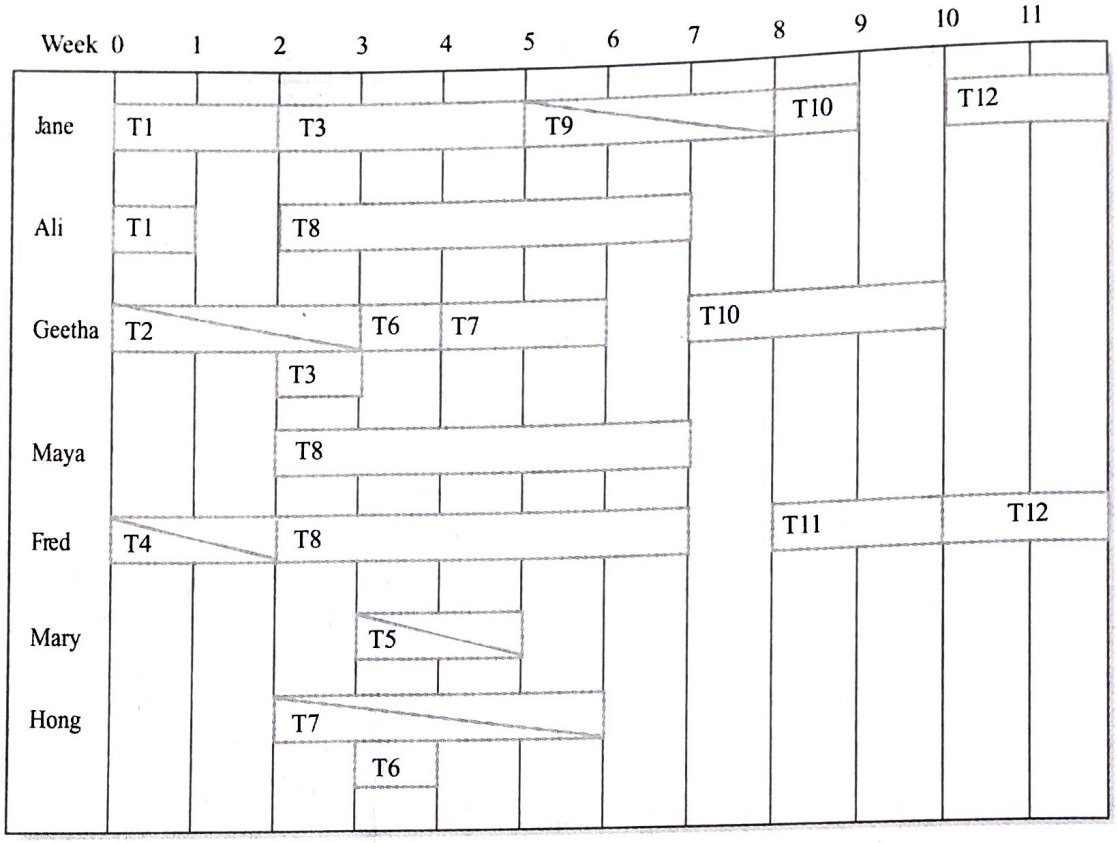

Figure 23.7 Staff allocation chart

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) is an excellent introduction to agile planning.

Agile development methods such as Scrum (Rubin 2013) and Extreme Programming (Beck and Andres 2004) 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 usually represents 2 to 4 weeks of work for the team.

I have already explained the Scrum approach to planning in Chapter 3, which is based on project backlogs and daily reviews of work to be done. It is primarily geared