8

Project Planning and Control

LEARNING OBJECTIVES

- To appreciate looking at project control from a system point of view
- To be aware of typical project situations, and ways in which projects can be successfully dealt with in such situations
- To understand how risks can be prevented from becoming problems
- To know techniques for the day-to-day planning and control of software development projects

In this chapter I try to reconcile the various approaches sketched in chapters 3--7. A taxonomy of software development projects is given, together with recommended management practices for dealing with such projects. The chapter also deals with risk management and some well-known techniques for project planning and control.

Software development projects differ widely. These differences are reflected in the ways in which these projects are organized and managed. For some projects, the budget is fixed and the goal of the project is to maximize the quality of the end product. For others, quality constraints are fixed in advance, and the goal is to produce effectively a system that meets those quality constraints. If the developing organization has considerable experience with the application domain and the requirements are fixed and stable, a tightly structured approach may yield a satisfactory solution. In applications with fuzzy requirements and little previous experience in the development team, a more agile approach may be desirable.

It is important to identify those project characteristics early on, because they will influence the way a project is organized, planned and controlled. In section 8.1, we will discuss project control from a systems point of view. This allows us to identify the major dimensions along which software development projects differ. These dimensions lead to a taxonomy of software development projects, which will be discussed in section 8.2. For each of the project categories distinguished, we will indicate how best to control the various entities identified in previous chapters. This type of assessment is to be done at the project planning stage.

This assessment links global risk categories to preferred control situations. Daily practice, however, is more complex. An actual project faces many risks, each of which has to be handled in turn. Even risks for which we hoped to have found an adequate solution, may turn into problems later on. Risk factors therefore have to be monitored, and contingency plans have to be developed. The early identification of risks and the development and carrying out of strategies to mitigate these risks is known as **risk management**. Risk management is discussed in section 8.3.

Software development projects consist of a number of interrelated tasks. Some of these will have to be handled sequentially (a module cannot be tested until it has been implemented), while others may be handled in parallel (different modules can be implemented concurrently). The dependencies between tasks can be depicted in a network from which a project schedule can be derived. These and similar tools for the micro-level planning and control of software development projects are discussed in section 8.4.

8.1 A Systems View of Project Control

In the preceding chapters, we discussed several entities that need to be controlled. During the execution of a software development project, each of these entities needs

to be monitored and assessed. From time to time, adjustments will have to be made. To be able to do so, we must know which entities can be varied, how they can be varied, and what the effect of adjustments is.

To this end, we will consider project control from a systems point of view. We now consider the software development project itself as a system. Project control may then be described in terms of:

- the system to be controlled, i.e. the software development project;
- the entity that controls the system, i.e. the project manager, his organization and the decision rules he uses;
- information which is used to guide the decision process. This information may come from two sources. It may either come from the system being controlled (such as a notice of technical problems with a certain component) or it may have a source outside the system (such as a request to shorten development time).

The variables that play a role in controlling a system may be categorized into three classes: irregular variables, goal variables, and control variables.

Irregular variables are those variables that are input to the system being controlled. Irregular variables cannot be varied by the entity that controls the system. Their values are determined by the system's environment. Examples of irregular variables are the computer experience of the user or the project staffing level.

An important precondition for effective control is knowledge of the project's goals. In developing software, various conflicting goals can be distinguished. One possible goal is to *minimize development time*. Since time is often pressing, this goal is not unusual. Another goal might be to *maximize efficiency*, i.e. development should be done as cheaply as possible. Optimal use of resources (mostly manpower) is then needed. Yet a third possible goal is to *maximize quality*. Each of these goals is possible, but they can be achieved only if it is known which goals are being pursued. These goals collectively make up the set of goal variables.

Finally, the decision process is guided by the set of control variables. Control variables are entities which can be manipulated by the project manager in order to achieve the goals set forth. Examples of possible control variables are the tools to be used, project organization, efficiency of the resulting software.

It is not possible to make a rigid separation between the various sets of variables. It depends on the situation at hand whether a particular variable should be taken as an irregular variable, goal variable, or control variable. If the requirements are stable and fixed, one may for instance try to control the project by employing adequate personnel and using a proper set of tools. For another project, manpower may be fixed and one may try to control the project by extending the delivery date, relaxing quality constraints, etc.

However, in order to be able to control a project, the different sets of variables must be known. It must be known where control is, and is not, possible. This is only

one prerequisite, though. In systems theory, the following conditions for effective control of a system are used:

- the controlling entity must know the goals of the system;
- the controlling entity must have sufficient control variety;
- the controlling entity must have information on the state, input and output of the system;
- the controlling entity must have a conceptual control model. It must know how and to what extent the different variables depend on and influence each other.

When all these conditions are met, control can be rational, in which case there is no uncertainty, since the controlling entity is completely informed about every relevant aspect. The control problem can then be structured and formalized. Daily practice of software development is different, though. There is insufficient room for control or the effect of control actions is not known. Control then becomes much more intuitive or primitive. It is based on intuition, experience, and rules of thumb.

The degree to which a software development project can be controlled increases as the control variety increases. This control variety is determined by the number of control variables and the degree to which they can be varied. As noticed before, the control variety is project dependent.

Controlling software development means that we must be able to measure both the project and the product. Measuring a project means that we must we able to assess progress. Measuring a product means that we must be able to determine the degree to which quality and functional requirements are being met.

Controlling software development projects implies that effective control actions are possible. Corrective actions may be required if progress is not sufficient or the software does not comply with its requirements. Effective control means that we know what the effect of control actions is. If progress is insufficient and we decide to allocate extra manpower, we must understand the impact of this extra manpower on the time schedule. If the quality of a certain component is less than required and we decide to allocate extra test time, we must know how much test time is required in order to achieve the desired quality.

In practice, controlling a software development project is not a rational process. The ideal systems theory situation is not met. There are a number of uncertainties which make managing such projects a challenging activity. Below, we will discuss a few idealized situations, based on the uncertainty of various relevant aspects.

8.2 A Taxonomy of Software Development Projects

In the preceding section, we identified several conditions that need to be satisfied in order to be able to control projects rationally. Since these conditions are often not met, we will have to rely on a different control mechanism in most cases. The

control mechanism best suited to any given situation obviously depends on relevant characteristics of the project at hand.

Based on an analysis of software development project characteristics that are important for project control, we will distinguish several project situations, and indicate how projects can successfully be controlled in these situations.

We will group project characteristics into three classes: product characteristics, process characteristics, and resource characteristics. From the point of view of project control, we are interested in the degree of *certainty* of those characteristics. For example, if we have clear and stable user requirements, product certainty is high. If part of the problem is to identify user requirements, or the user requirements frequently change during the development project, product certainty is low.

If product certainty is high, control can be quite rational, insofar as it depends on product characteristics. Since we know what the product is supposed to accomplish, we may check compliance with the requirements and execute corrective actions if needed. If product certainty is low, this is not feasible. We either do not know what we are aiming at, or the target is constantly moving. It is only reasonable to expect that control will be different in those cases.

For the present discussion, we are interested only in project characteristics that may differ between projects. Characteristics common to most or all of software development projects, such as the fact that they involve teamwork, will not lead to different control paradigms.

We will furthermore combine the characteristics from each of the three categories identified above, into one metric, the certainty of the corresponding category. This leaves us with three dimensions along which software development projects may differ:

- **Product certainty** Product certainty is largely determined by two factors: whether or not user requirements are clearly specified, as regards both functionality and quality, and the volatility of those user requirements. Other product characteristics are felt to have a lesser impact on our understanding of what the end-product should accomplish.

- **Process certainty** The degree of (development) process certainty is determined by such factors as: the possibility of redirecting the development process, the degree to which the process can be measured and the knowledge we have about the effect of control actions, and the degree to which new, unknown tools are being used.

- **Resource certainty** The major determinant here is the availability of the appropriate qualified personnel.

If we allow each of these certainty factors to take one of two values (high and low), we get eight control situations, although some of them are not very realistic. If we have little or no certainty about the software to be developed, we can hardly expect to be certain about the process to be followed and the resources needed to accomplish our

goals. Similarly, if we do not know how to carry out the development process, we also do not know which resources are needed.

This leaves us with four archetypal situations, as depicted in figure 8.1. Below, we will discuss each of these control situations in turn. In doing so, we will pay attention to the following aspects of those control situations:

- the kind of control problem;
- the primary goals to be set in controlling the project;
- the coordination mechanism to be used;
- the development strategy, or process model, to be applied;
- the way and degree to which cost can be estimated.

	Realization	Allocation	Design	Exploration
Product certainty	high	high	high	low
Process certainty	high	high	low	low
Resource certainty	high	low	low	low

Figure 8.1 Four archetypal control situations

- **Realization problem** If the requirements are known and stable, it is known how the software is to be developed, there is sufficient control variety, the effect of control actions is known, and sufficient resources are available, we find ourselves in an ideal situation, a situation not often encountered in our field. The main emphasis will be on realization: how can we, given the requirements, achieve our goal in the most effective way? As for the development strategy, we may use some linear process model. Feedback to earlier phases, as in the waterfall model, is needed only for verification and validation activities.

  To coordinate activities in a project of this type, we may use direct supervision. Work output can be standardized, since the end result is known. Similarly, the work processes and worker skills can be fixed in advance. There will thus be little need for control variety as far as these variables are concerned.

  Management can be done effectively through a separation style. The work to be done is fixed through rules and procedures. Management can allocate tasks and check their proper execution.

As for cost estimation, we may successfully use one of the more formalized cost models. Alternatively, experts in the domain may give a reliable estimate. A cost estimation thus obtained can be used to guard the project's progress and yields a target to be achieved.

- **Allocation problem** This situation differs from the previous one in that there is uncertainty as regards the resources. The major problem then becomes one of the availability of personnel. Controlling a project of this kind tends to become one of controlling capacity. The crucial questions become: How do we get the project staffed? How do we achieve the desired end-product with limited means?

  According to Mintzberg, one has to try to standardize the process as far as possible in this case. This makes it easier to move personnel between tasks. Guidelines and procedures may be used to describe how the various tasks have to be carried out.

  As regards the development strategy, we may again opt for the waterfall model. We may either contract out the work to be done, or try to acquire the right type and amount of qualified personnel.

  As for cost estimation, either some cost estimation model or expert estimates can be used. Since there is uncertainty as regards resources, there is a need for sensitivity analyses in order to gain insight into such questions as: What will happen to the total cost and development time if we allocate three designers of level A rather than four designers of level B?

- **Design problem** If the requirements are fixed and stable, but we do not know how to carry out the process, nor which resources to employ, the problem is one of design. Note that the adjective *design* refers to the design of the project, not the design of the software. We have to answer such questions as: which milestones are to be identified, which documents must be delivered and when, what personnel must be allocated, how will responsibilities be assigned?

  In this situation, we have insufficient knowledge of the effect of allocating extra personnel, other tools, different methods and techniques. The main problem then becomes one of controlling the development process.

  In Mintzberg's classification, this can best be pursued through standardization of work outputs. Since the output is fixed, control should be done through the process and the resources. The effect of such control actions is not sufficiently known, however.

  In order to make a project of this kind manageable, one needs overcapacity. As far as the process is concerned, this necessitates margins in development time and budget. Keeping extra personnel is not feasible, in general.

  In these situations, we will need frequently to measure progress towards the project's goals in order to allow for timely adjustments. Therefore, we may want

to go from a linear development model to an incremental one. This preference will increase as the uncertainty increases.

Cost estimation will have to rely on past experience. We will usually not have enough data to use one of the more formalized cost estimation models. In this situation too, we will need sensitivity analyses. This need will be more pressing than in the previous situation, since the uncertainty is greater. The project manager will be interested in the sensitivity of cost estimates to certain cost drivers. He might be interested in such questions as: what will happen to the development schedule if two extra analysts are assigned to this project, or: what will the effect be on the total cost if we shorten the development time by x days? By viewing cost estimation in this way, the manager will gain insight to, and increase his feeling for, possible solution strategies.

- **Exploration problem** If the product certainty, process certainty and resource certainty are all low, we get the most difficult control situation.

  Because of these uncertainties, the work will be exploratory in nature. This situation does not fit a coordination mechanism based on standardization. In a situation as complex and uncertain as this one, coordination can best be achieved through mutual adjustment. The structure is one of adhocracy. Experts from various disciplines work together to achieve some as yet unspecified goal.

  A critical success factor in these cases is the commitment of all people involved. Work cannot be split up into neat tasks. Flexibility in work patterns and work contents is important. Adherence to a strict budget cannot be enforced upon the team from above. The team members must commit themselves to the project. Management has to place emphasis on their relations with the team members.

  Controlling a project of this kind is a difficult and challenging activity. To make a project of this kind manageable, our goal will be to maximize output, given the resources available to the project. This maximization may concern the quality of the product, or its functionality, or both.

  Since requirements are not precisely known, some agile approach is appropriate as a process model. The larger the uncertainty, the more often we will have to check whether we are still on the right track. Thus, some development strategy involving many small steps and frequent user feedback is to be used. Cost estimation using some formalized model clearly is not feasible in these circumstances. The use of such models presupposes that we know enough of the project at hand to be able to compare it with previous projects. Such is not the case, though.

  We may rely on expert judgments to achieve a rough cost estimate. Such a cost estimate, however, cannot and should not be used as a fixed anchor point as to when the project should be finished and how much it may cost. There are simply too many uncertainties involved. Rather, it provides us with some

guidance as to the magnitude of the project. Based on this estimate, effort and time can be allocated for the project, for instance to produce a certain number of prototypes, a feasibility study, a pilot implementation of part of the product, or to start a certain number of time boxes. The hope is that in time the uncertainties will diminish sufficiently so that the project shifts to one of the other situations.

The four control situations discussed above are once more depicted in figure 8.2, together with a short characterization of the various control aspects discussed above.

For big projects, it may be effective to use different control mechanisms at the macro and micro level, respectively (Karlström and Runeson, 2005). At the macro level, management may have to coordinate the work of different teams, and report to higher management. This may require an approach in which explicit stages and corresponding milestones are distinguished. At the level of a small subteam though, one may still apply agile methods to control the day-to-day work.

By taking the different control aspects into account during the planning stage of a software development project, we can tailor the project's management to the situation at hand. In doing so, we recognize that software development projects are not all alike. Neglecting those project-specific characteristics is likely to result in project failures, failures that have often been reported upon in the literature, but equally often remain hidden from the public at large.

8.3 Risk Management

Risk management is project management for adults
Tim Lister

In the previous section, we identified global risk categories and tied them to preferred control situations. In this section, the emphasis is on individual risks and their management *during* project execution. In some sense too, the discussion below takes a more realistic point of view, in that we also consider adverse situations such as unrealistically tight schedules and design gold plating.

Potential risks of a project must be identified as early as possible. It is rather naive to suppose that a software project will run smoothly from start to finish. It won't. We should identify the risks of a software project early on and provide measures to deal with them. Doing so is not a sign of unwarranted pessimism. Rather, it is a sign of wisdom.

In software development, we tend to ignore risks. We assume an optimistic scenario under all circumstances and we do not reserve funds for dealing with risks. We rely on heroics when chaos sets in. If risks are identified at all, their severity is often underestimated, especially by observers higher in the hierarchy. A designer may have noticed that a certain subsystem poses serious performance problems. His manager assumes that the problem can be solved. His manager's manager assumes the problem *has* been solved.

Problem type	Realization	Allocation	Design	Exploration
Product certainty	high	high	high	low
Process certainty	high	high	low	low
Resource certainty	high	low	low	low
Primary goal in control	Optimize resource usage Efficiency and schedule	Acquisition, training of personnel	Control of the process	Maximize result Lower risks
Coordination, Management style	Standardization of product, process, and resources Hierarchy, separation style	Standardization of product and process	Standardization of process	Mutual adjustment Commitment Relation style
Development strategy	Waterfall	Waterfall	Incremental	Incremental Prototyping Agile
Cost estimation	Models Guard process	Models Sensitivity analysis	Expert estimate Sensitivity analysis	Expert estimate Risk analysis Provide guidance

Figure 8.2 Four control situations (*After: F.J. Heemstra*, How much does software cost, *Kluwer Bedrijfswetenschappen, 1989.*)

A risk is a possible future negative event that may affect the success of an effort. So, a risk is not a problem, yet. It may become one, though, and risk management is concerned with *preventing* risks from becoming problems. Some common examples of risks and ways to deal with them, are:

- Requirements may be unstable, immature, unrealistic, or excessive. If we merely list the requirements and start to realize the system in a linear development mode, it is likely that a lot of rework will be needed. This results in schedule and budget overruns, since this rework was not planned. If the requirements volatility is identified as a major risk, an evolutionary development strategy can be chosen. This situation fits the exploration-problem category as identified in the previous section.

- If there is little or no user involvement during the early development stages, a real danger is that the system will not meet user needs. If this is identified as a risk, it can be mitigated, e.g. by having users participate in design reviews.
- If the project involves different or complex domains, the spread of application knowledge within the project team may be an issue. Recognizing this risk may result in timely attention and resources for a training program for team members.
- If the project involves more than one development site, communication problems may arise. A common way to deal with this is to pay attention to socialization issues, for instance by scheduling site visits.

At the project planning stage, risks are identified and handled. A risk management strategy involves the following steps:

1. Identify the risk factors. There are many possible risk factors. Each organization may develop its own checklist of such factors. The top ten risk factors from (Boehm, 1989) are listed in figure 8.3.
2. Determine the risk exposure. For each risk, we have to determine the probability p that it will actually occur and the effect E (e.g. in dollars or loss of man months) that it will have on the project. The risk exposure then equals $p \times E$.
3. Develop strategies to mitigate the risks. Usually, this will only be done for the N risks that have the highest risk exposure, or for those risks whose exposure exceeds some threshold α.

   There are three general strategies to mitigate risks: avoidance, transfer, and acceptance. We may avoid risks by taking precautions so that they will not occur: buy more memory, assign more people, provide for a training program for team members, and the like. We may transfer risks by looking for another solution, such as a prototyping approach to handle unstable requirements. Finally, we may accept risks. In the latter case, we have to provide for a contingency plan, to be invoked when the risk does become a problem.
4. Handle risks. Risk factors must be monitored. For some risks, the avoidance or transfer actions may succeed, and those risks will never become a problem. We may be less lucky for those risks that we decided up front not to handle. Also, some of our actions may turn out to be less successful, and risks that we hoped to have handled adequately may become a problem after all. Finally, project characteristics will change over time, and so will the risks. Risk management thus is a cyclic process, and occasionally risks must be handled by re-assessing the project, invoking a contingency plan, or even a transfer to crisis mode.

Wallace and Keil (2004) give a useful categorization of risk factors. They distinguish four types of risk (see also figure 8.4):

Risk	Description
Personnel shortfall	May manifest itself in a variety of ways, such as inexperience with the domain, tools or development techniques to be used, personnel turnover, loss of critical team members, or the mere size of a team.
Unrealistic schedule/budget	Estimates may be unrealistic with respect to the requirements.
Wrong functionality	May have a variety of causes, such as an imperfect understanding of the customer needs, the complexity of communication with the client, insufficient domain knowledge of the developers and designers.
Wrong user interface	In certain situations, the user-friendliness of the interface is critical to its success.
Gold plating	Developers may wish to develop 'nice' features not asked for by the customer.
Requirements volatility	If many requirements change during development, the amount of rework increases.
Bad external components	The quality or functionality of externally supplied components may be below what is required for this project.
Bad external tasks	Subcontractors may deliver inadequate products, or the skills obtained from outside the team may be inadequate.
Real-time shortfalls	The real-time performance of (parts of) the system may be inadequate.
Capability shortfalls	An unstable environment or new or untried technology pose a risk to the development schedule.

Figure 8.3 Top ten risk factors

- C1: Risks related to customers and users. Examples include a lack of user participation, conflicts between users, or a user organization resisting change. Part of this can be mitigated through an agile approach. But equally often, such risks are beyond the project manager's control.

- C2: Risks that have to do with the scope of the project and its requirements. Various factors from figure 8.3 fall into this category: wrong functionality, gold plating, requirements volatility. Project managers should be able to control this type of risk.

- C3: Risks that concern the execution of the project: staffing, methodology, planning, control. Factors like personnel shortfall and an unrealistic schedule or budget belong to this category. Again, project managers should be able to control these risks.

- C4: Risks that result from changes in the environment, such as changes in the organization in which the system is to be embedded, or dependencies on outsourcing partners. Project managers often have few means to control these risks.

relative importance	Level of control: low	Level of control: high
low	customers and users (C1)	scope and requirements (C2)
high	environment (C4)	execution (C3)

Figure 8.4 Risk categories

From a study of a large number of projects, Wallace and Keil (2004) found that risk categories C2 and C3 affect project outcomes most. They also found that execution risks (C3) are much more important in explaining process outcome that scope or requirements risks. This would suggest an ordering amongst the types of risks that managers had better pay attention to: first C3, then C2, and finally C4 and C1.

When you return to figure 8.3 after having studied the remainder of this book, you will note that many of the risk factors listed are extensively addressed in various chapters. These risk factors surface as cost drivers in cost estimation models, the quest for user involvement in requirements engineering and design, the attention for process models like prototyping and XP, and so on.

As Tom Gilb says: 'If you don't actively attack the risks, they will actively attack you' (Gilb, 1988, p. 72).

8.4 Techniques for Project Planning and Control

A project consists of a series of activities. We may graphically depict the project and its constituent activities by a **work breakdown structure** (WBS). The WBS reflects the decomposition of a project into subtasks down to a level needed for effective planning and control. Figure 8.3 contains a very simple example of a work breakdown structure for a software development project. The activities depicted at the leaves of the work breakdown structure correspond to unit tasks, while the higher-level nodes constitute composite tasks. We will assume that each activity has a well-defined beginning and end that is indicated by a milestone, a scheduled event for which some person is held accountable and which is used to measure and control progress. The end of an activity is often a deliverable, such as a design document, while the start of an activity is often triggered by the end of some other activity.

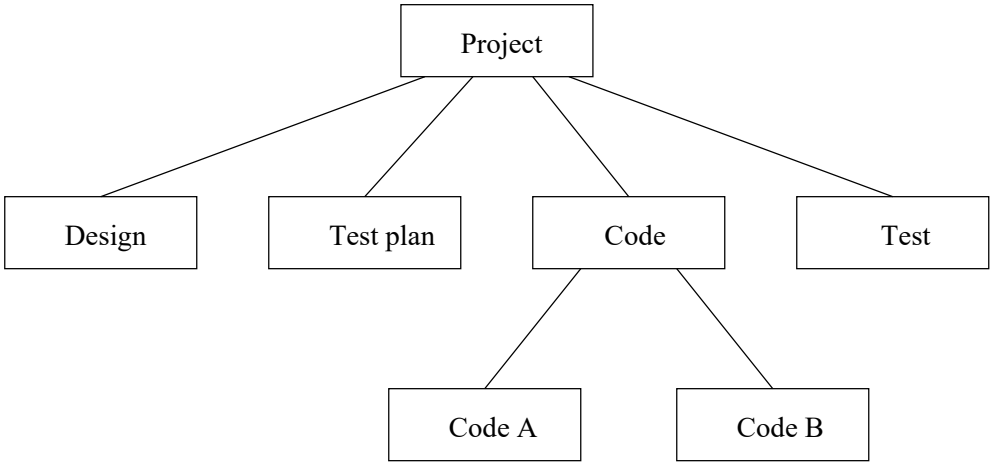

Figure 8.5 Simple work breakdown structure for a software development project

Activities usually consume resources, such as people or computer time, and always have a certain duration. Activities must often be executed in a specific order. For example, we can not test a module before it is coded. This type of relation between tasks can be expressed as constraints. Usually, the constraints concern temporal relations between activities. Such constraints are also called precedence relations. Project planning involves the scheduling of all activities such that the constraints are satisfied and resource limits are not exceeded. Several techniques are available to support this scheduling task.

The activities from the simple WBS of a software development project, together with their duration and temporal constraints, are given in figure 8.6. Note that figure 8.6 contains more information on temporal relations than is given in the WBS. Though the left-to-right reading of the WBS suggests a certain time ordering, it does not give the precise precedence relations between activities.

Activity	Duration	Constraints
Design	10	--
Test plan	5	Design finished
Code A	10	Design finished
Code B	5	Design finished
Test	10	Code finished, Test plan finished

Figure 8.6 Activities, their duration and temporal constraints

The set of activities and their constraints can also be depicted in a network. For our example, this network is given in figure 8.7. The nodes in the network denote activities. This type of network is therefore known as an 'activity-on-node' network. Each node also carries a weight, the duration of the corresponding activity. An arrow from node A to node B indicates that activity A has to be finished before activity B can start.

These network diagrams are often termed **PERT charts**. PERT is an acronym for Program Evaluation and Review Technique. PERT charts were developed and first used successfully in the management of the Polaris missile program in the 1950s. While the original PERT technique was concerned solely with the time span of activities and their interrelations, subsequent developments have led to a variety of techniques that accommodate an increasing number of project factors.

From the PERT chart we may compute the earliest possible point in time at which the project can be completed. Let us assume that the network has a unique start node B and end node E. If there is more than one node with in-degree 0 (i.e. having no predecessors in the network), a new start node B is created with outgoing edges to all nodes having in-degree 0. This new node B gets a zero weight (duration). A similar procedure is followed to create the end node E if there is more than one node having out-degree 0.

We next label each node i in the network with an ordered pair of numbers (S_i, F_i). S_i and F_i denote the earliest possible time at which activity i can start and finish, respectively. The algorithm for doing so involves a breadth-first search of the network (cf. (Boehm, 1981)):

1. The start node B is labeled $(0, D_B)$, where D_B is the duration of activity B;

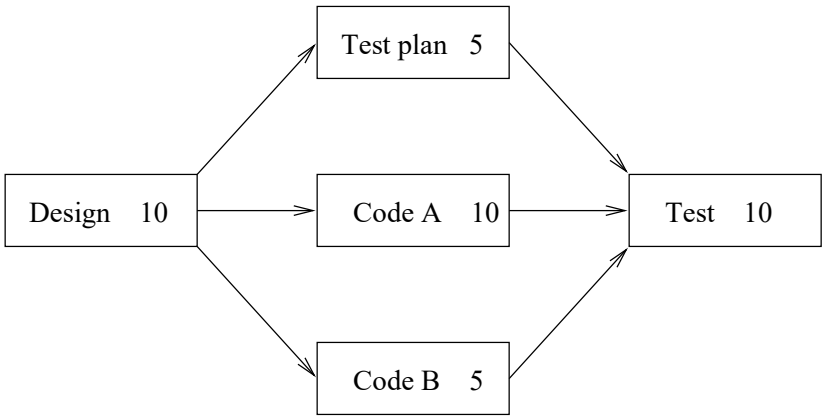

Figure 8.7 Example of a PERT chart

2. For all unlabeled nodes whose predecessors are all labeled nodes, the earliest possible starting time is the latest finishing time of all the predecessor nodes:

$$S_N = \max_{i \in P(N)} F_i$$

where $P(N)$ is the set of predecessor nodes of N.

The corresponding finishing time is $F_N = S_N + D_N$, where D_N is the duration of activity N.

Node N is labeled as (S_N, F_N).

3. Repeat Step 2 until all nodes have been labeled.

The earliest possible finishing time of the whole project now equals F_E, E being the end node of the network.

We may subsequently compute the latest point in time at which activity L should finish: for each node N,

$$L_N = \min_{i \in Q(N)} S_i$$

where $Q(N)$ is the set of successor nodes of N.

The results of this computation can be graphically presented in a **Gantt chart** (these charts are named after their inventor). In a Gantt chart, the time span of each activity is depicted by the length of a segment drawn on an adjacent calendar. The Gantt chart of our software development example is given in figure 8.8. The gray areas show slack (or float) times of activities. It indicates that the corresponding

activity may consume more than its estimated time, or start later than the earliest possible starting time, without affecting the total duration of the project. For each activity N, the corresponding segment in the Gantt chart starts at time S_N and ends at L_N.

Activities without slack time are on a **critical path**. If activities on a critical path are delayed, the total project gets delayed as well. Note that there always is at least one sequence of activities that constitutes a critical path.

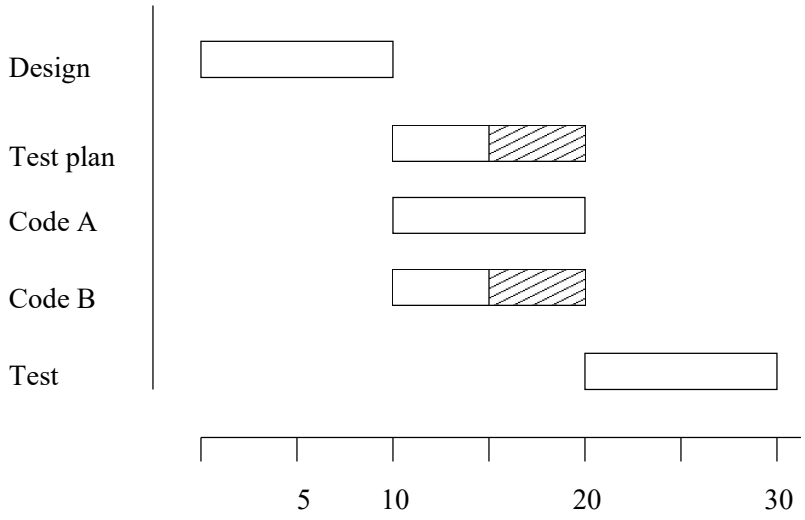

Figure 8.8 Example of a Gantt chart

In an 'activity-on-node' network, the activities are depicted as nodes, while the arrows denote precedence relations between activities. Alternatively, we may depict a set of interrelated activities in an 'activity-on-arrow' network. In an activity-on-arrow network, the arrows denote activities, while the nodes represent the completion of milestone events. Figure 8.9 depicts the example as an activity-on-arrow network. The latter representation is intuitively appealing, especially if the length of an arrow reflects the duration of the corresponding activity. Note that this type of network may have to contain dummy activities which are not needed in the activity-on-node network. These dummy activities represent synchronization of interrelated activities. In our example, dummy activities (arrows) are needed to make sure that the activity *test* is not started until the activities *test plan*, *code A* and *code B* have all been completed.

The PERT technique has evolved considerably since its inception 50 years ago. For example, as well as expressing a constraint that activity B may start only after an activity A has ended, we may also specify that activity B may only start after

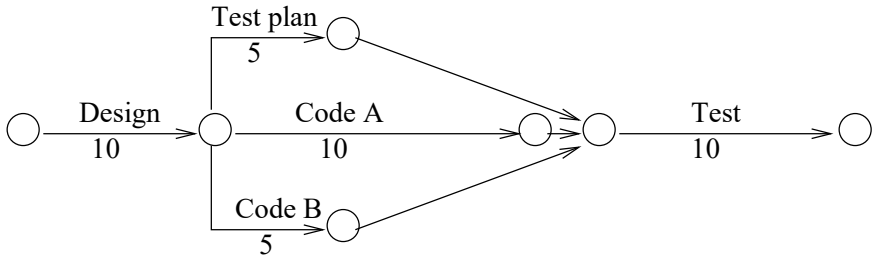

Figure 8.9 An activity-on-arrow network

activity A has started. We may also extend the technique so that it handles resource constraints. For instance, if we have only one programmer available, the Gantt chart of figure 8.6 would not work, since it assumes that coding of modules A and B is done in parallel. The PERT technique may even be extended further to allow for sensitivity analysis. By allowing so-called 'what-if' questions ('what if we allocate three designers rather than four', 'what if coding module A takes two months rather than one') we get a feeling for the sensitivity of a schedule to certain variations in resource levels, schedule overruns, and the like.

Critical Path Method -- CPM -- is, as the name suggests, a technique very similar to PERT and developed at around the same time.

In our discussion, we presented a Gantt chart as a graphical visualization of a schedule that results from network analysis. Actually, we may use a Gantt chart as a scheduling mechanism in its own right. We may simply list all activities and indicate their earliest starting time and latest ending time on the calendar. Gantt charts by themselves, however, do not carry information on dependencies between activities. This makes it hard to adjust schedules, for instance when a certain activity slips. As far as planning goes, we therefore prefer the use of Gantt charts as a means to visualize the result of network analysis.

Using the information contained in the Gantt chart and knowledge of personnel resources required for each activity, we may establish a personnel plan indicating how many people are required in each unit of time. Since people costs are a major part of project expenditures, this personnel plan provides a direct means to plan project expenditures.

When the project is under way, its control is based on monitoring the project's progress and expenditures. Time spent per activity per project member can be recorded on time cards. These time cards are the basis for determining cumulative effort and expenditure. These cumulative data can be compared with the planned levels of effort and expenditure. In order to properly judge whether the project is still on track, management needs progress information as well. The most common way to provide this is via milestone reports: activities cannot be considered completed until

a proper report has been produced and accepted.

The Gantt chart provides a very direct means to compare actual project status with the project schedule. Schedule slippage shows itself immediately. Slippage of activities on a critical path then necessitates prompt management action: renegotiation of the schedule, the project's deliverables, or both. Note that schedule slippage is a sneaky affair; projects get behind one day at a time. Note also that project schedules should at any point in time reflect the true project. An accepted change necessitates reconsideration of the schedule.

8.5 Summary

In this chapter we looked at project control from a systems point of view and gained insight into how different kinds of projects can be managed and controlled. We identified four archetypal situations, which demand different process models, coordination mechanisms and management styles.

Real projects face many risks, and it is a wise project manager who pays attention to them early on. A risk is a possible future negative event that may affect success. It is not a problem yet, but it may become one. Risk management is concerned with *preventing* risks from becoming problems. It involves the following steps:

1. Identify the risk factors.

2. Determine the risk exposure, i.e. the probability that a risk will happen, multiplied by its cost.

3. Develop strategies to mitigate risks, especially those with a high risk exposure. Risks may be avoided (e.g. by hiring more people), transferred (e.g. by choosing a different development strategy), or accepted.

4. Handle the risks: monitor risk factors and take action when needed.

In section 8.4, we focused on the planning and control of activities within a project. By depicting the set of activities and their temporal relations in a graph, techniques like PERT offer simple yet powerful means to schedule and control these activities (see, for example, (Boehm, 1981)).

8.6 Further Reading

Some general software project management sources are: (Boehm, 1981), (Brooks, 1995), (Humphrey, 1997b) and (Royce, 1998). Highsmith (2004) focusses on agile project management, while Boehm and Turner (2003) and Karlström and Runeson (2005) discuss how to combine agile and plan-based approaches.

The discussion in section 8.2 is based on (Heemstra, 1989).

(Boehm, 1989) gives a good overview of software risk management. Risk management experiences are reported on in (Software, 1997a). The risk categories discussed in section 8.3 stem from (Wallace and Keil, 2004). Pfleeger (2000) compares software risk management with risk management in other disciplines.

Exercises

1. List the conditions for effective systems control.
2. Is the waterfall approach suitable for a realization-type problem? If so, why?
3. Is the waterfall approach suitable for an exploration-type problem? If so, why?
4. What is risk management?
5. How can risks be mitigated?
6. Rephrase the cost drivers of the COCOMO cost estimation model as risk factors.
7. What is a work breakdown structure?
8. What is a PERT chart?
9. What is a Gantt chart?
10. ♠ Classify a project you have been involved in with respect to product certainty, process certainty, and resource certainty. Which of the archetypal situations sketched in section 8.2 best fits this project? In what ways did actual project control differ from that suggested for the situation identified? Can you explain possible differences?
11. ♡ Consider the patient planning system mentioned in exercise 3.12. Suppose the project team consists of several analysts and two members from the hospital staff. The analysts have a lot of experience in the design of planning systems, though not for hospitals. As a manager of this team, which coordination mechanism and management style would you opt for?
12. ♡ Discuss the pros and cons of a hierarchical as well as a matrix team organization for the patient planning project.
13. ♠ Consider a project you have been involved in. Identify the major irregular, control, and goal variables for this project. In what ways did the control variables influence project control?
14. ♡ Suppose one of your team members is dissatisfied with his situation. He has been involved in similar projects for several years now. You have assigned

him these jobs because he was performing so well. Discuss possible actions to prevent this employee from leaving the organization.

15. ♡ Why is planning (i.e., the activity) more important than the plan (the document)?

16. ♡ Suppose you are the manager of a project that is getting seriously behind schedule. Your team is having severe problems with testing a particular subsystem. Your client is pressing you to deliver the system on time. How would you handle this situation? How would you handle the situation if you were a member of the team and your manager was not paying serious attention to your signals?