2

Introduction to Software Engineering Management

LEARNING OBJECTIVES

- To be aware of the contents of a project plan
- To understand the major dimensions along which a software development project is controlled

Software development projects often involve several people for a prolonged period of time. Large projects may even range over several years and involve hundreds of people. Such projects must be carefully planned and controlled. The main aspects that deserve the continuous attention of project managers are introduced in this chapter.

It is not easy to complete successfully a software development project. This book mainly deals with technical aspects of software development: design, specification, implementation and testing of software systems. As we learn to control these aspects better, we will also learn to satisfy our customer's demands better. The organizational and managerial aspects of software development projects are at least as important as the technical aspects, though.

Before we embark on a discussion of these organizational and managerial aspects, let us first pay some attention to the boundaries of a software development project as they are drawn in this book.

A software development project is usually not started in complete isolation. There are other projects within the organization that this particular project needs to be tuned to, priorities between projects have to be decided upon, etc. The term **information planning** is often used to refer to this meta-project planning process.

Also in a more technical sense, projects are not started in isolation. To increase interoperability between systems, overall guidelines regarding, e.g., the use of certain standards, data interchange formats, security policies, web page layout and the like are laid down for the whole organization and imposed on every project. In product line development, the architecture of the product line guides the development of individual products.

These project exceeding rules result in a set of boundary conditions for each project, much like the zoning regulations set the conditions for a building project. Establishing these company-wide rules is a problem on its own, and will not be addressed here. (We will, however, pay ample attention to some issues which generally surpass the boundaries of individual software development projects, such as configuration control, quality assurance and product line development.)

Also in a more technical sense, software is not generally developed in isolation. In most cases, software is not written from scratch. It must interface with existing software, extend existing software, use existing subroutine libraries, build upon an existing framework, and so on.

In some sense, the notion of a 'software development project' is a misnomer. We do not just develop software, we develop systems. Broadly speaking, a system transforms inputs into outputs. Software is an important ingredient of the systems we develop, but it is by no means the only ingredient. The technical and user documentation, the hardware, the procedures that govern the use of the system, and even the people using the software, may be considered as part of that same system.

Consider for example a system for library automation. The system will contain

various software components, such as a database component to store information on books and customers and an interaction component to process user requests. As well as the development of these components, attention should be paid to matters like:

- techniques to identify books electronically, such as a barcode scheme;
- the selection and acquisition of special hardware both for scanning those identifications and for producing identifications for new books;
- setting up a scheme to provide all books with the new identification code;
- instruction of library employees to handle the new types of equipment (training material and courses, operating procedures, and the like);
- production of user-friendly documentation for the library customers.
- web-accessibility issues, such as whether the catalog can be browsed, or books can be reserved on-line

Whenever the notion 'software development project' is used in the following, it should be understood in this wider sense. This is graphically illustrated in figure 2.1.

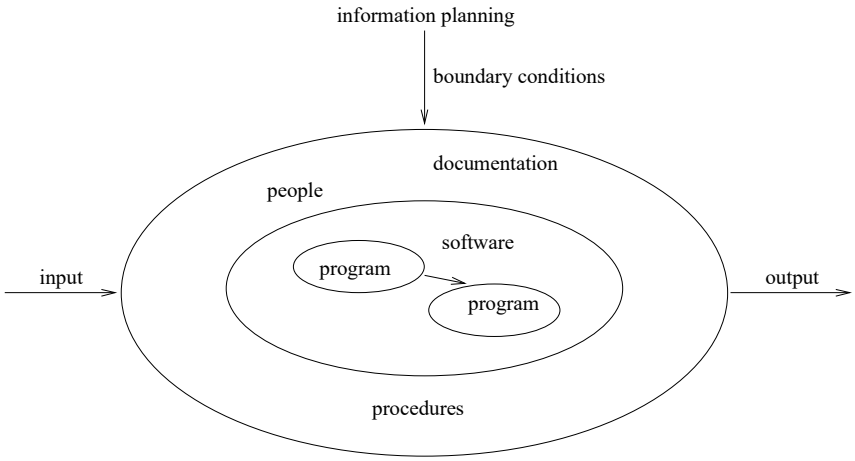

Figure 2.1 The systems view of a software development project

Thus, our systems encompass a number of components. In a narrow sense, the software component itself may also consist of a number of interacting components. These latter components correspond to programs as we know them from introductory

computer science textbooks. In general, a software development project results in a set of components which collectively provide us with the desired functionality.

Given a project's boundary conditions, a software development project may get started. Planning the project is the very first step to be undertaken. Part of this planning process is to identify the project characteristics and their impact on the development process. The result of the planning phase is laid down in a document, the **project plan**, which aims to provide a clear picture of the project to both the customers and the development team. The contents of the project plan are discussed in section 2.1.

During the execution of the project, a number of elements have to be managed: time, information, organization, quality, and money (see section 2.2). Each of these elements is further elaborated upon in a separate chapter.

2.1 Planning a Software Development Project

Before we embark on a software development project, it has to be carefully planned. This entails, amongst other things, an assessment of project properties that may affect the development process. A number of properties, however, will not be sufficiently well understood until the requirements engineering phase has ended. Like many other aspects of a software development project, planning is not a one-shot activity. Rather, it is highly dynamic in nature. The project plan can serve as a guide during the project.

The amount of upfront planning depends on characteristics of the problem at hand. In highly explorative projects, where requirements are largely unknown at the start, too rigorous early planning can be stifling and may increase the probability of major failures. A strict 'plan the work and work the plan' attitude does not work in these circumstances. Rather, such projects call for a nominal early planning and a management style that encourages responding to change. This is reflected in the contents and size of the project plan. Chapters 3 and 8 further discuss differences between the so-called agile and planning-driven approaches to software development.

The major constituents of a project plan are:

1. **Introduction** In the introduction to the project plan, the background and history of the project are given, together with its aims, the project deliverables, the names of the persons responsible, and a summary of the project.

2. **Process model** In chapter 1, we introduced a simple life cycle model in order to discuss the various activities to be dealt with in a software development project. There exist many variations of this process model, some of which are discussed in chapter 3. For each project, one has to decide upon the exact process model to be followed: which activities are being undertaken, which milestones can be identified, how do we ascertain whether those milestones are reached, and which are the critical paths.

Different types of projects have different characteristics, and so call for different process models.

3. **Organization of the project** The relationship of the project to other entities and the organization of the project itself are dealt with under this heading. The project will have a relationship with the user organization, the parent organization, and possibly with other organizations.

   The prospective users will from time to time be involved in the project. The project plan has to state which information, services, resources and facilities are to be provided by the users and when these are to be provided.

   Within the project team, various roles can be identified: project manager, tester, programmer, analyst, etc. One has to clearly delineate these roles and identify the responsibilities of each of them. If there are gaps in the knowledge required to fulfill any of these roles, the training and education needed to fill these gaps have to be identified. Different forms of team organization are discussed in chapter 5.

4. **Standards, guidelines, procedures** Software projects are big projects. Usually, a lot of people are involved. A strong working discipline is therefore needed, in which each person involved follows the standards, guidelines and procedures agreed upon. Besides being stated on paper, many of these can be supported or enforced by tools. Of extreme importance are clear agreements about documentation: when is documentation to be delivered, how is the quality of the documentation to be assessed, how does one ensure that the documentation is kept up-to-date?

   To a large extent, these standards and procedures will be described in separate documents, such as the Configuration Control Plan or the Quality Assurance Plan.

5. **Management activities** Management activities are guided by goals and priorities set for the project. For example, management will have to submit regular reports on the status and progress of the project. It will also have to follow certain priorities in balancing requirements, schedule and cost.

6. **Risks** Potential risks have to be identified as early as possible. There will always be risks: hardware may not be delivered on time, qualified personnel may not be available when required, critical information may be lacking when it is needed, and so on. It is rather naive to suppose that a software development project runs smoothly. Even in well-established fields like construction, there is always something that goes wrong. One should diagnose the risks of a software project early on, and provide measures to deal with them; see also chapter 8.

   The more uncertain various aspects of the project are, the larger the risks.

7. **Staffing** At different points in time, the project will require different amounts of personnel, with different skills. The start, duration, amount and expertise of personnel categories are listed under this heading.

8. **Methods and techniques** Under this heading, the methods and techniques to be used during requirements engineering, design, implementation and testing are given. Typically, the way version and configuration control for software components is dealt with is described here too. A large proportion of the technical documentation will be produced during these phases. One thus has to state how this documentation will be taken care of.

   The necessary test environment and test equipment is described. During testing, considerable pressure will normally be put on the test equipment. Therefore, this activity has to be planned carefully. The order in which components are integrated and tested has to be stated explicitly. Also, the procedures to be followed during acceptance testing, i.e. the testing under user supervision, have to be given. Testing will be discussed in chapter 13.

9. **Quality assurance** Which organization and procedures will be used to assure that the software being developed meets the quality requirements stated? The many aspects of a Quality Assurance Plan may also be dealt with in a separate document. The topic of quality assurance is discussed in chapter 6.

10. **Work packages** Larger projects must be broken down into activities, manageable pieces that can be allocated to individual team members. Each of these activities has to be identified in the project plan. The hierarchical decomposition of the project is depicted in a work breakdown structure (see also section 8.4).

11. **Resources** During the project, many resources are needed. The hardware, CPU-cycles and tools needed to support the project are listed under this entry. One should also indicate the personnel needed for the various process phases.

12. **Budget and schedule** The total budget for the project has to be allocated to the various activities as indicated in the work breakdown structure. The activities also have to be scheduled in time, e.g. using a PERT chart (see section 8.4). The way in which resources and other expenditures are tracked is also indicated under this heading. The topic of cost and time estimation will be dealt with extensively in chapter 7.

13. **Changes** It has been stated before that changes are inevitable. One has to ensure that these changes are dealt with in an orderly way. One thus needs clear procedures on how proposed changes will be handled. If the process is agile, every iteration involves changes, and these are dealt with in a lightweight manner. In fact, they are not seen as changes anymore. In more heavyweight processes, each proposed change must be registered and reviewed. When a

change request has been approved, its impact (cost) has to be estimated. Finally, the change has to be incorporated into the project. Changes that are entered via the back door lead to badly structured code, insufficient documentation and cost and time overruns. Since changes lead to different versions of both documentation and code, the procedures to be followed in dealing with such changes are often handled in the context of a Configuration Control Plan.

14. **Delivery** The procedures to be followed in handing over the system to the customer must be stated.

The project plan aims to provide a clear picture of the project to both the customers and the project team. If the objectives are not clear, they will not be achieved.

Despite careful planning, surprises will still crop up during the project. However, careful planning early on leads to fewer surprises and makes one less vulnerable to these surprises. The project plan addresses a number of questions which anticipate possible future events. It gives orderly procedures for dealing with those events, so that justifiable decisions can be reached.

2.2 Controlling a Software Development Project

After a project plan has been drawn up and approved, the execution of the project may start. During the project, control has to be exerted along the following dimensions:

- time,
- information,
- organization,
- quality,
- money.

Progress of a software development project (the **time** aspect) is hard to measure. Before the proposed system has been finished, there is only a (large) pile of paper. Utterances such as '90% of the code has been written' should be taken with a pinch of salt. A much too rosy picture of the actual state of affairs is usually given. The phased approach introduced in chapter 1, and its variants, aim at providing the manager with an instrument to measure and control progress. The time needed to build a system is obviously related to the size of the system, and thus to the total manpower required. Larger systems require more time to develop, although we may try to shorten development time by allocating more personnel. Part of the control problem for software development projects is to trade off time against people. Adding more people to shorten development time does not come for free. The more people that are involved, the more time will be needed for coordination and communication. After a certain point, adding more people actually lengthens the development time.

Part of the time control problem is phrased in Brooks' Law: 'Adding people to a late project only makes it later'. We will come back to this issue in the chapter on cost estimation.

The **information** that has to be managed, above all, is the documentation. Besides technical and user documentation, this also entails documentation on the project itself. Documentation concerning the project includes such things as: the current state of affairs, changes that have been agreed upon, and decisions that have been made. This type of documentation can best be handled in the context of configuration management. In agile projects, less attention is given to documentation during development. Necessary knowledge is tacit, it resides in the heads of the people involved. But here too, once the system is ready and handed over to its customers, documentation has to be provided.

All members of the development team must understand their role in the team and what is expected of them. It is very important that these expectations are clear to all people involved. Unspoken and unclear expectations lead to situations in which individual team members set their own goals, either consciously or unconsciously. These **organizational** aspects deserve the continuous attention of the project manager. Secondly, the organization of a team and the coordination of the people involved will, at least partly, depend upon characteristics of the project and its environment. This dependence has to be recognized and taken into account when setting up a project team.

The **quality** aspect is of paramount importance. Customers are not satisfied with the purely technical solutions offered by computer specialists. They want systems that fit their real needs. The quality requirements for software and its development often conflict with one another. At architecture time, quality requirements are balanced in a dialog with all stakeholders involved. During a project we will have to assess whether or not the quality requirements are being met. This quality assessment has to occur on a regular basis, so that timely actions can be undertaken. Quality is not an add-on feature, it has to be built in.

Controlling expenses (the **money** aspect) largely means controlling labor costs. Though the cost of hardware and tools cannot be ignored, these can usually be estimated fairly precisely early in the project. Moreover, these are usually much less of an issue than personnel costs.

Estimating the cost of software thus means that we must estimate the manpower required to build the software. The manpower needed is very much dependent on the size of the software, for instance measured as the amount of code to be delivered. Many other factors, though, influence this cost or, alternatively, the productivity with which the software can be produced. A well-balanced team with experienced people will be much more productive than a newly-formed team with inexperienced people. Extremely strict quality constraints, such as very high reliability or a very fast response time, may also severely reduce productivity.

A number of models have been proposed that try to quantify the effect of those different cost drivers on the manpower required (see chapter 7). Rather than

estimating the size first, and then its cost, one may also set a cost threshold first, and work incrementally, first on the most pressing user requirements and, if time allows, on less pressing ones. Or one may agree on a first threshold, and decide whether more money will be spent when this threshold is reached. These incremental approaches to cost estimation fit in well with agile project development.

Software development is a very labor-intensive process. One of our hopes is that better tools and the increased use of those tools will lead to a significant increase in productivity and, consequently, a significant decrease in the cost involved in developing software. A second way, to increase productivity dramatically, is to use software rather than build it yourself. Both these topics will be discussed in chapters to follow. As these trends continue, software development starts to become a capital-intensive activity, rather than a labor-intensive one (Wegner, 1984).

Continuous assessment of the project with respect to these control aspects is of the utmost importance and will from time to time lead to adjustments in time, cost, organization, information, or quality, or some combination thereof. Project management is a very dynamic activity.

In order to be able to adequately control a project, we need quantitative data which is collected while the project is being executed. For instance, data about errors discovered during unit testing may help us in estimating further test effort needed. Data about the time and effort spent up to a specific point will guide us in re-estimating the schedule and cost. To measure is to know.

These data are also valuable in a post-mortem evaluation of the project. In a post-mortem evaluation we assess the present project in order to improve our performance on projects yet to come: what have we done wrong, what have we learned, what needs to be done differently on the next project?

Unfortunately, in practice very little hard data is ever gathered, let alone retained for later use. Most software development organizations have little insight into what they are doing. They tend to operate in a somewhat chaotic way, especially when facing a crisis. By identifying key factors that affect the controllability of the software development process, we may find ways to improve on it. This topic is further treated in chapter 6, where we discuss the Software Capability Maturity Model.

2.3 Summary

This chapter provides an introduction to the management of software engineering projects.

Before we embark on a software development project, it has to be carefully planned. This planning process results in a document, the project plan, which provides a clear picture of the project to both the customers and the project team.

Once the project plan has been drawn up and the project has started, its execution must be controlled. We identified five entities that require our continuous attention for project control:

- Time: How do we assess progress towards the project's goals? Usually, some phased approach is followed which aims to provide management with a means to measure and control progress.
- Information: How do we handle the documents that are produced in the course of a project? In planning-based development, maintaining the integrity of the set of documents and handling all change requests require careful procedures.
- Organization: How do we organize the project team and coordinate the activities of team members?
- Quality: How do we define and assess quality requirements for both the development process and the resulting product?
- Money: How do we estimate the cost of a project? These costs are to a large extent determined by the size of the software.

Each of these controlling aspects is further elaborated upon in a separate chapter (chapters 3--7). The various dimensions of project control will then be reconciled in chapter 8.

Exercises

1. In what sense is the phrase 'software development project' a misnomer?
2. What are the major constituents of a project plan?
3. List five dimensions along which a software development project has to be controlled.
4. How may software development become a capital-intensive activity, rather than a labor-intensive one?
5. ♠ Consider a software development project you have been involved in. Did the project have a project plan? Did the project plan address the issues listed in section 2.1? If some of these issues were not addressed, do you think it would have helped the project if they had been?
6. ♡ Do you think quantitative project data are important? In what way can they contribute to project planning?
7. ♡ How would a project plan for an agile project differ from that of a planning-driven project?
8. ♠ Consider once again a software development project you have been involved in. To what extent were any environmental issues such as user training and working procedures adequately dealt with in the project?

9. ♡ A program written for personal use imposes rather less stringent requirements than a product that is also to be used by other people. According to (Brooks, 1995), the latter may require three times as much effort. Discuss possible reasons for this considerable increase in cost.