Project Planning in Software Engineering

Before starting a software project, it is essential to determine the tasks to be performed and properly manage allocation of tasks among individuals involved in the software development. Hence, planning is important as it results in effective software development. Project planning is an organized and integrated management process, which focuses on activities required for successful completion of the project. It prevents obstacles that arise in the project such as changes in projects or organization's objectives, non-availability of resources, and so on. Project planning also helps in better utilization of resources and optimal usage of the allotted time for a project. The other objectives of project planning are listed below.

- It defines the roles and responsibilities of the project management team members.
- It ensures that the project management team works according to the business objectives.
- It checks feasibility of the schedule and user requirements.
- It determines project constraints.

Several individuals help in planning the project. These include senior management and project management team. Senior management is responsible for employing team members and providing resources required for the project. The project management team, which generally includes project managers and developers, is responsible for planning, determining, and tracking the activities of the project. Table lists the tasks performed by individuals involved in the software project.

Tasks of Individuals involved in Software Project

Senior Management	Project Management Team
• Approves the project, employ personnel, and provides resources required for the project. • Reviews project plan to ensure that it accomplishes the business objectives. • Resolves conflicts among the team members. • Considers risks that may affect the project so that appropriate measures can be taken to avoid them.	• Reviews the project plan and implements procedures for completing the project. • Manages all project activities. • Prepares budget and resource allocation plans. • Helps in resource distribution, project management, issue resolution, and so on. • Understands project objectives and finds ways to accomplish the objectives. • Devotes appropriate time and effort to achieve the expected results. • Selects methods and tools for the project.

Project planning should be effective so that the project begins with well-defined tasks. Effective project planning helps to minimize the additional costs incurred on the project while it is in progress. For effective project planning, some principles are followed. These principles are listed below.

- **Planning is necessary:** Planning should be done before a project begins. For effective planning, objectives and schedules should be clear and understandable.
- **Risk analysis:** Before starting the project, senior management and the project management team should consider the risks that may affect the project. For example, the user may desire changes in requirements while the project is in progress. In such a case, the estimation of time and cost should be done according to those requirements (new requirements).
- **Tracking of project plan:** Once the project plan is prepared, it should be tracked and modified accordingly.
- **Meet quality standards and produce quality deliverables:** The project plan should identify processes by which the project management team can ensure quality in software. Based on the process selected for ensuring quality, the time and cost for the project is estimated.

Problem-Based Estimation

'nes of code and function points were described as measures from which productivity metrics can be computed
C and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size'
ı element of the software and (2) as baseline metrics collected from past projects and used in conjunction with
ffort projections

- **Description of flexibility to accommodate changes:** The result of project planning is recorded in the form of a project plan, which should allow new changes to be accommodated when the project is in progress.

Project planning comprises project purpose, project scope, project planning process, and project plan. This information is essential for effective project planning and to assist project management team in accomplishing user requirements.

Project Purpose

Software project is carried out to accomplish a specific purpose, which is classified into two categories, namely, project objectives and business objectives. The commonly followed project objectives are listed below.

- **Meet user requirements:** Develop the project according to the user requirements after understanding them.
- **Meet schedule deadlines:** Complete the project milestones as described in the project plan on time in order to complete the project according to the schedule.
- **Be within budget:** Manage the overall project cost so that the project is within the allocated budget.
- **Produce quality deliverables:** Ensure that quality is considered for accuracy and overall performance of the project.

Business Software Engineering

Business objectives ensure that the organizational objectives and requirements are accomplished in the project. Generally, these objectives are related to business process improvements, customer satisfaction, and quality improvements. The commonly followed business objectives are listed below.

- **Evaluate processes:** Evaluate the business processes and make changes when and where required as the project progresses.
- **Renew policies and processes:** Provide flexibility to renew the policies and processes of the organization in order to perform the tasks effectively.
- **Keep the project on schedule:** Reduce the downtime (period when no work is done) factors such as unavailability of resources during software development.
- **Improve software:** Use suitable processes in order to develop software that meets organizational requirements and provides competitive advantage to the organization.

Project Scope

With the help of user requirements, the project management team determines the scope of the project before the project begins. This scope provides a detailed description of functions, features, constraints, and interfaces of the software that are to be considered. Functions describe the tasks that the software is expected to perform. Features describe the attributes required in the software as per the user requirements. Constraints describe the limitations imposed on software by hardware, memory, and so on. Interfaces describe the interaction of software components (like modules and functions) with each other. Project scope also considers software performance, which in turn depends on its processing capability and response time required to produce the output.

Once the project scope is determined, it is important to properly understand it in order to develop software according to the user requirements. After this, project cost and duration are estimated. Lf the project scope is not determined on time, the project may not be completed within the specified schedule. Project scope describes the following information.

- The elements included and excluded in the project
- The processes and entities
- The functions and features required in software according to the user requirements.

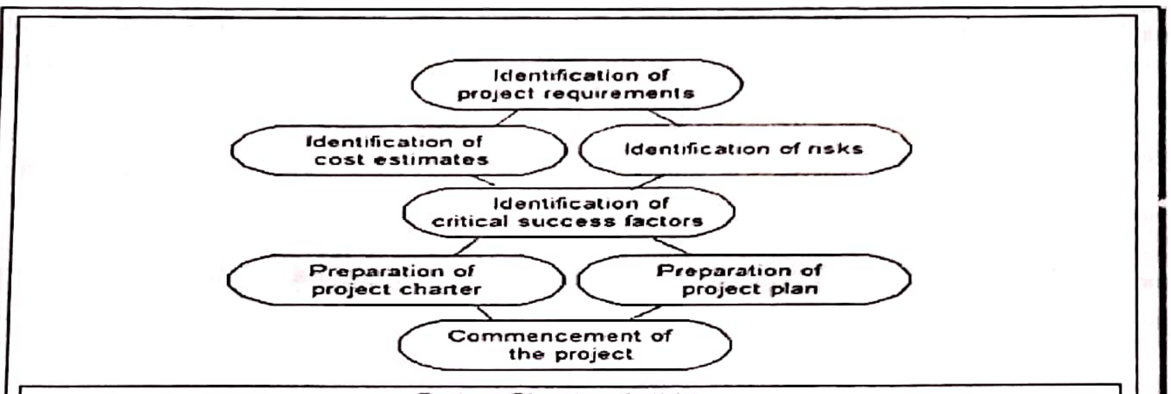
Note that the project management and senior management team should communicate with the users to understand their requirements and develop software according to those requirements and expected functionalities.

Project Planning Process

The project planning process involves a set of interrelated activities followed in an orderly manner to implement user requirements in software and includes the description of a series of project planning activities and individual(s) responsible for performing these activities. In addition, the project planning process comprises the following.

1. Objectives and scope of the project
2. Techniques used to perform project planning
3. Effort (in time) of individuals involved in project
4. Project schedule and milestones
5. Resources required for the project
6. Risks associated with the project.

Project planning process comprises several activities, which are essential for carrying out a project systematically. These activities refer to the series of tasks performed over a period of time for developing the software. These activities include estimation of time, effort, and resources required and risks associated with the project.


Identification of project requirements
Identification of cost estimates
Identification of risks
Identification of critical success factors
Preparation of project charter
Preparation of project plan
Commencement of the project

Project Planning Activities

Project planning process consists of the following activities.

- **Identification of project requirements:** Before starting a project, it is essential to identify the project requirements as identification of project requirements helps in performing the activities in a systematic manner. These requirements comprise information such as project scope, data and functionality required in the software, and roles of the project management team members.

- **Identification of cost estimates:** Along with the estimation of effort and time, it is necessary to estimate the cost that is to be incurred on a project. The cost estimation includes the cost of hardware, network connections, and the cost required for the maintenance of hardware components. In addition, cost is estimated for the individuals involved in the project.
- **Identification of risks:** Risks are unexpected events that have an adverse effect on the project. Software project involves several risks (like technical risks and business risks) that affect the project schedule and increase the cost of the project. Identifying risks before a project begins helps in understanding their probable extent of impact on the project.
- **Identification of critical success factors:** For making a project successful, critical success factors are followed. These factors refer to the conditions that ensure greater chances of success of a project. Generally, these factors include support from management, appropriate budget, appropriate schedule, and skilled software engineers.

- **Preparation of project charter:** A project charter provides a brief description of the project scope, quality, time, cost, and resource constraints as described during project planning. It is prepared by the management for approval from the sponsor of the project.
- **Preparation of project plan:** A project plan provides information about the resources that are available for the project, individuals involved in the project, and the schedule according to which the project is to be carried out.
- **Commencement of the project:** Once the project planning is complete and resources are assigned to team members, the software project commences.

Once the project objectives and business objectives are determined, the project end date is fixed. The project management team prepares the project plan and schedule according to the end date of the project. After analyzing the project plan, the project manager communicates the project plan and end date to the senior management. The progress of the project is reported to the management from time to time. Similarly, when the project is complete, senior management is informed about it. In case of delay in completing the project, the project plan is re-analyzed and corrective actions are taken to complete the project. The project is tracked regularly and when the project plan is modified, the senior management is informed.

Project Plan

As stated earlier, a project plan stores the outcome of project planning. It provides information about the end date, milestones, activities, and deliverables of the project. In addition, it describes the responsibilities of the project management team and the resources required for the project. It also includes the description of hardware and software (such as compilers and interfaces) and lists the methods and standards to be used. These methods and standards include algorithms, tools, review techniques, design language, programming language, and testing techniques.

A project plan helps a project manager to understand, monitor, and control the development of software project. This plan is used as a means of communication between the users and project management team. There are various advantages associated with a project plan, some of which are listed below.

- It ensures that software is developed according to the user requirements, objectives, and scope of the project.
- It identifies the role of each project management team member involved in the project.
- It monitors the progress of the project according to the project plan.
- It determines the available resources and the activities to be performed during software development.
- It provides an overview to management about the costs of the software project, which are estimated during project planning.

Note that there are differences in the contents of two project plans depending on the kind of project and user requirements. Atypical project plan is divided into the following sections.

1. **Introduction:** Describes the objectives of the project and provides information about the constraints that affect the software project.
2. **Project organization:** Describes the responsibilities assigned to the project management team members for completing the project.
3. **Risk analysis:** Describes the risks that can possibly arise during software development as well as explains how to assess and reduce the effect of risks.
4. **Resource requirements:** Specifies the hardware and software required to carry out the software project. Cost estimation is done according to these resource requirements.
5. **Workbreakdown:** Describes the activities into which the project is divided. It also describes the milestones and deliverables of the project activities.
6. **Project schedule:** Specifies the dependencies of activities on each other. Based on this, the time required by the project management team members to complete the project activities is estimated.

tables to develop cost and effort projections.

have a number of characteristics in common.

In addition to these sections, there are several plans that may be a part of or 'linked to a project plan. These plans include quality assurance plan, verification and validation plan, configuration management plan, maintenance plan, and staffing plan.

Quality Assurance Plan

The quality assurance plan describes the strategies and methods that are to be followed to accomplish the following objectives.

1. Ensure that the project is managed, developed, and implemented in an organized way.
2. Ensure that project deliverables are of acceptable quality before they are delivered to the user.

Verification and Validation Plan

The verification and validation plan describes the approach, resources and schedule used for system validation. The verification and validation plan, which comprises the following sections.

1. **General information:** Provides description of the purpose, scope, system overview, project references, acronyms and abbreviations, and points of contact. Purpose describes the procedure to verify and validate the components of the system. Scope provides information about the procedures to verify and validate as they relate to the project. System overview provides information about the organization responsible for the project and other information such as system name, system category, operational status of the system, and system environment. Project references provide the list of references used for the preparation of the verification and validation plan. Acronyms and abbreviations provide a list of terms used in the document. Points of contact provide information to users when they require assistance from organization for problems such as troubleshooting and so on.

2. Reviews and walkthroughs: Provides information about the schedule and procedures. Schedule describes the end date of milestones of the project. Procedures describe the tasks associated with reviews and walkthroughs. Each team member reviews the document for errors and consistency with the project requirements. For walkthroughs, the project management team checks the project for correctness according to software requirements specification (SRS).

3. System test plan and procedures: Provides information about the system test strategy, database integration, and platform system integration. System test strategy provides an overview of the components required for integration of the database and ensures that the application runs on at least two specific platforms. Database integration procedure describes how database is connected to the Graphical User Interface (GUI).Platform system integration procedure is performed on different operating systems to test the platform.

4. Acceptance test and preparation for delivery: Provides information about procedure, acceptance criteria, and installation procedure. Procedure describes how acceptance testing is to be performed on the software to verify its usability as required. Acceptance criteria describes that software will be accepted only if all the components, features and functions are tested including the system integration testing. In addition, acceptance criteria checks whether the software accomplishes user expectations such as its ability to operate on several platforms. Installation procedure describes the steps of how to install the software according to the operating system being used.

Configuration Management Plan

The configuration management plan defines the process, which is used for making changes to the project scope. Generally, the configuration management plan is concerned with redefining the existing objectives of the project and deliverables (software products that are delivered to the user after completion of software development).

maintenance plan specifies the resources and processes required for making the software operational after installation. Sometimes, the project management team (or software development team) does not carry out the ask of maintenance. In such a case, a separate team known as software maintenance team performs the task of software maintenance. The maintenance plan, which comprises the sections listed below.

1. Introduction and background: Provides a description of software to be maintained and the services required for it. It also specifies the scope of maintenance activities that are to be performed.

2. Budget: Specifies the budget required for carrying out software maintenance and operational activities.

3. Roles and responsibilities: Specifies the roles and responsibilities of the team members associated with the software maintenance and operation. It also describes the skills required to perform maintenance and operational activities. In addition to software maintenance team, software maintenance comprises user support, user training, and support staff.

4. Performance measures and reporting: Identifies the performance measures required for carrying out software maintenance. It also describes how measures required for enhancing the performance of services (for the software) are recorded and reported.

5. Management approach: Identifies the methodologies that are required for establishing maintenance priorities of the projects. For this purpose, the management either refers to the existing methodologies or identifies new methodologies. Management approach also describes how users are involved in software maintenance and operations activities as well as how users and project management team communicate with each other.

6. Documentation strategies: Provides a description of the documentation that is prepared for user reference. Generally, documentation includes reports, information about problems occurring in software, error messages, and the system documentation.

7. Training: Provides information about the training activities.

8. Acceptance: Defines a point of agreement between the project management team and software maintenance team after the completion of implementation and transition activities. Once the agreement has been made, the software maintenance begins.

Staffing Plan

The staffing plan describes the number of individuals required for a project. It includes selecting and assigning tasks to the project management team members. It provides information about appropriate skills required to perform the tasks to produce the project deliverables and manage the project. In addition, it provides information of resources such as tools, equipment, and processes used by the project management team.

Staff planning is performed by a staff planner, who is responsible for determining the individuals available for the project. Other responsibilities of a staff planner are listed below.

1. The staff planner determines individuals, who can be from existing staff, staff on contract, or newly employed staff. It is important for the staff planner to know the structure of the organization to determine the availability of staff.

2. The staff planner determines the skills required to execute the tasks mentioned in the project schedule an task plan. In case staff with required skills is not available, staff planner informs the project manager about the requirements.

enance Plan

maintenance plan specifies the resources and processes required for making the software operational after installation. Sometimes, the project management team (or software development team) does not carry out the sk of maintenance. In such a case, a separate team known as software maintenance team performs the task of ftware maintenance. The maintenance plan, which comprises the sections listed below.

1. Introduction and background: Provides a description of software to be maintained and the services required for it. It also specifies the scope of maintenance activities that are to be performed.

2. Budget: Specifies the budget required for carrying out software maintenance and operational activities.

3. Roles and responsibilities: Specifies the roles and responsibilities of the team members associated with the software maintenance and operation. It also describes the skills required to perform maintenance and operational activities. In addition to software maintenance team, software maintenance comprises user support, user training, and support staff.

4. Performance measures and reporting: Identifies the performance measures required for carrying out software maintenance. It also describes how measures required for enhancing the performance of services (for the software) are recorded and reported.

5. Management approach: Identifies the methodologies that are required for establishing maintenance priorities of the projects. For this purpose, the management either refers to the existing methodologies or identifies new methodologies. Management approach also describes how users are involved in software maintenance and operations activities as well as how users and project management team communicate with each other.

6. Documentation strategies: Provides a description of the documentation that is prepared for user reference. Generally, documentation includes reports, information about problems occurring in software, error messages, and the system documentation.

7. Training: Provides information about the training activities.

8. Acceptance: Defines a point of agreement between the project management team and software maintenance team after the completion of implementation and transition activities. Once the agreement has been made, the software maintenance begins.

Staffing Plan

The staffing plan describes the number of individuals required for a project. It includes selecting and assigning tasks to the project management team members. It provides information about appropriate skills required to perform the tasks to produce the project deliverables and manage the project. In addition, it provides information of resources such as tools, equipment, and processes used by the project management team.

Staff planning is performed by a staff planner, who is responsible for determining the individuals available for the project. Other responsibilities of a staff planner are listed below.

1. The staff planner determines individuals, who can be from existing staff, staff on contract, or newly employed staff. It is important for the staff planner to know the structure of the organization to determine the availability of staff.

2. The staff planner determines the skills required to execute the tasks mentioned in the project schedule and task plan. In case staff with required skills is not available, staff planner informs the project manager about the requirements.

Problem-Based Estimation

Lines of code and function points were described as measures from which productivity metrics can be comput
LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "s
each element of the software and (2) as baseline metrics collected from past projects and used in conjunctio
ation variables to develop cost and effort projections.

are distinct estimation techniques. Yet both have a number of characteristics in c
a bounded statement of software scope and from this statement at
that can each be estimated individually. LOC or FP (the
ively, the planner may choose another compone
ropriate estimatio
verall

The staff planner ensures that the required staff with required skills is available at the right time. For this urpose, the staff planner plans the availability of staff after the project schedule is fixed. For example, at the nitial stage of a project, staff may consist of a project manager and a few software engineers whereas during software development, staff consists of software designers as well as the software developers.

4. The staff planner defines roles and responsibilities of the project management team members so that they can communicate and coordinate with each other according to the tasks assigned to them. Note that the project management team can be further broken down into sub-teams depending on the size and complexity of the project.

The staffing plan comprises the following sections.

1. General information: Provides information such as name of the project and project manager who is responsible for the project. In addition, it specifies the start and end dates of the project.

2. Skills assessment: Provides information, which is required for assessment of skills. This information includes the knowledge, skill, and ability of team members who are required to achieve the objectives of the project. In addition, it specifies the number of team members required for the project.

3. Staffing profile: Describes the profile of the staff required for the project. The profile includes calendar time, individuals involved, and level of commitment. Calendar time specifies the period of time such as month or quarter for which individuals are required to complete the project. Individuals who are involved in the project have specific designations such as project manager and the developer. Level of commitment is the utilization rate of individuals such as work performed on full-time and part-time basis.

4. Organization chart: Describes the organization of project management team members. In addition, it includes information such as name, designation, and role of each team member.

Problem-Based Estimation

Lines of code and function points were described as measures from which productivity metrics can be computed LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the planner may choose another component for sizing such as classes or objects, changes, or business processes affected.

Baseline productivity metrics (e.g., LOC/pm or FP/pm) are then applied to the appropriate estimation variable, and cost or effort for the function is derived. Function estimates are combined to produce an overall estimate for the entire project.

It is important to note, however, that there is often substantial scatter in productivity metrics for an organization. making the use of a single baseline productivity metric suspect. In general, LOC/pm or FP/pm averages should be computed by project domain. That is, projects should be grouped by team size, application area, complexity, and other relevant parameters. Local domain averages should then be computed. When a new project is estimated, it should first be allocated to a domain, and then the appropriate domain average for productivity should be used in generating the estimate.

The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of the partitioning. When LOC is used as the estimation variable, decomposition is absolutely essential and is often taken to considerable levels of detail. The following decomposition approach has been adapted from Phillips :

```
define product scope;
identify functions by decomposing scope;
do while functions remain
select a functionj
assign all functions to subfunctions list;
do while subfunctions remain
select subfunctionk
if subfunctionk  resembles subfunctiond described in a historical data base

then note historical cost, effort, size (LOC or FP) data for subfunctiond;

adjust historical cost, effort, size data based on any differences;

use adjusted cost, effort, size data to derive partial estimate, Ep;

project estimate = sum of {Ep};

else if cost, effort, size (LOC or FP) for subfunctionk can be estimated

then derive partial estimate, Ep;

project estimate = sum of {Ep};

else subdivide subfunctionk into smaller subfunctions;

add these to subfunctions list;

endif

endif
```

This decomposition approach assumes that all functions can be decomposed into subfunctions that will resemble entries in a historical data base. If this is not the case, then another sizing approach must be applied. The greater the degree of partitioning, the more likely reasonably accurate estimates of LOC can be developed.

For FP estimates, decomposition works differently. Rather than focusing on function, each of the information domain characteristics—inputs, outputs, data files, inquiries, and external interfaçes—as well as the 14 complexity adjustment are estimated. The resultant estimates can then be used to derive a FP value that can be tied to past data and used to generate an estimate.

Regardless of the estimation variable that is used, the project planner begins by estimating a range of values for each function or information domain value. Using historical data or (when all else fails) intuition, the planner estimates an optimistic, most likely, and pessimistic size value for each function or count for each information domain value. An implicit indication of the degree of uncertainty is provided

when a range of values is specified.

A three-point or expected value can then be computed. The expected value for the estimation variable (size), S. can be computed as a weighted average of the optimistic (sopt), most likely (sm), and pessimistic (spess) estimates. For example,

$$S = (sopt + 4sm + spess)/6$$

gives heaviest credence to the "most likely" estimate and follows a beta probability distribution. We assume that there is a very small probability the actual size result will fall outside the optimistic or pessimistic values.

Once the expected value for the estimation variable has been determined, historical LOC or FP productivity data are applied.

Estimation: The process approximating a value that can be used even if the data may be incomplete or unstable is referred to as estimation.

Problem based estimation:

Begins with a statement of scope.

The software is decomposed into problem functions.

Estimating FP or LOC.

Combine those estimates and produce an overall estimate.

Process based estimation:

The functions of the software are identified.

The framework is formulated.

Estimate effort to complete each software function.

Apply average labor rates, compute the total cost and compare the estimates.

do

nddo

This decomposition approach assumes that all functions can be decomposed into subfunctions that will resemble entries in a historical data base. If this is not the case, then another sizing approach must be applied. The greater the degree of partitioning, the more likely reasonably accurate estimates of LOC can be developed.

For FP estimates, decomposition works differently. Rather than focusing on function, each of the information domain characteristics—inputs, outputs, data files, inquiries, and external interfaçes—as well as the 14 complexity adjustment are estimated. The resultant estimates can then be used to derive a FP value that can be tied to past data and used to generate an estimate.

Regardless of the estimation variable that is used, the project planner begins by estimating a range of values for each function or information domain value. Using historical data or (when all else fails) intuition, the planner estimates an optimistic, most likely, and pessimistic size value for each function or count for each information domain value. An implicit indication of the degree of uncertainty is provided

when a range of values is specified.

A three-point or expected value can then be computed. The expected value for the estimation variable (size), S. can be computed as a weighted average of the optimistic (sopt), most likely (sm), and pessimistic (spess) estimates. For example,

$$S = (sopt + 4sm + spess)/6$$

gives heaviest credence to the "most likely" estimate and follows a beta probability distribution. We assume that there is a very small probability the actual size result will fall outside the optimistic or pessimistic values.

Once the expected value for the estimation variable has been determined, historical LOC or FP productivity data are applied.

Estimation: The process approximating a value that can be used even if the data may be incomplete or unstable is referred to as estimation.

Problem based estimation:

Begins with a statement of scope.

The software is decomposed into problem functions.

Estimating FP or LOC.

Combine those estimates and produce an overall estimate.

Process based estimation:

The functions of the software are identified.

The framework is formulated.

Estimate effort to complete each software function.

Apply average labor rates, compute the total cost and compare the estimates.

ware Engineerig-Process-Based Estimation

most common technique for estimating a project is to base the estimate on the process that will be used. That the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is stimated.

Like the problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of software process activities must be performed for each function. Functions and related software process activities may be represented as part of a table .

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table. Average labour rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labour rate will vary for each task. Senior staff heavily involved in early activities are generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Costs and effort for each function and software process activity are computed as the last step. If process-based estimation is performed independently of LOC or FP estimation, we now have two or three estimates for cost and effort that may be compared and reconciled. If both sets of estimates show reasonable agreement, there is good reason to believe that the estimates are reliable. If, on the other hand, the results of these decomposition techniques show little agreement, further investigation and analysis must be conducted.

Process-Based Estimation

The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated. Like the problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of software process activities must be performed for each function. Functions and related software process activities may be represented as part of a table similar to the one presented.

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table. Average labour rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labour rate will vary for each task. Senior staff heavily involved in early activities is generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Problem-Based Estimation

Lines of code and function points were described as measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections. Problem-Based Estimation LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the plannermay choose another component for sizing such as classes or objects, changes, or business.

ware Engineerig-Process-Based Estimation

most common technique for estimating a project is to base the estimate on the process that will be used. That the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is stimated.

Like the problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of software process activities must be performed for each function. Functions and related software process activities may be represented as part of a table .

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table. Average labour rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labour rate will vary for each task. Senior staff heavily involved in early activities are generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Costs and effort for each function and software process activity are computed as the last step. If process-based estimation is performed independently of LOC or FP estimation, we now have two or three estimates for cost and effort that may be compared and reconciled. If both sets of estimates show reasonable agreement, there is good reason to believe that the estimates are reliable. If, on the other hand, the results of these decomposition techniques show little agreement, further investigation and analysis must be conducted.

Process-Based Estimation

The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated. Like the problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of software process activities must be performed for each function. Functions and related software process activities may be represented as part of a table similar to the one presented.

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table. Average labour rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labour rate will vary for each task. Senior staff heavily involved in early activities is generally more expensive than junior staff involved in later design tasks, code generation, and early testing.

Problem-Based Estimation

Lines of code and function points were described as measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections. Problem-Based Estimation LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the plannermay choose another component for sizing such as classes or objects, changes or business.

The availability of historical information has a strong influence on estimation risk. By looking back, we can emulate things that worked and improve areas where problems arose. When comprehensive software metrics (Chapter 22) are available for past projects, estimates can be made with greater assurance, schedules can be established to avoid past difficulties, and overall risk is reduced.

"It is the mark of an instructed mind to rest satisfied with the degree of precision that the nature of the subject admits, and not to seek exactness when only an approximation of the truth is possible."

Aristotle

2

Estimation risk is measured by the degree of uncertainty in the quantitative estimates established for resources, cost, and schedule. If project scope is poorly understood or project requirements are subject to change, uncertainty and estimation risk become dangerously high. The planner, and more importantly, the customer should recognize that variability in software requirements means instability in cost and schedule.

However, a project manager should not become obsessive about estimation. Modern software engineering approaches (e.g., incremental process models) take an iterative view of development. In such approaches, it is possible—although not always politically acceptable—to revisit the estimate (as more information is known) and revise it when the customer makes changes to requirements.

objective

23.2 THE PROJECT PLANNING PROCESS

ADVICE

The more you know, the better you estimate. Therefore, update your estimates as the project progresses.

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost, and schedule. In addition, estimates should attempt to define best-case and worst-case scenarios so that project outcomes can be bounded. Although there is an inherent degree of uncertainty, the software team embarks on a plan that has been established as a consequence of planning tasks. Therefore, the plan must be adapted and updated as the project proceeds. In the following sections, each of the activities associated with software project planning is discussed.

TASK SET

Task Set for Project Planning

1. Establish project scope ✓
2. Determine feasibility ✓
3. Analyze risks (Chapter 25) ✓
4. Define required resources
   a. Determine human resources required
   b. Define reusable software resources
   c. Identify environmental resources
5. Estimate cost and effort
   a. Decompose the problem
   b. Develop two or more estimates using size, function points, process tasks, or use-cases
   c. Reconcile the estimates
6. Develop a project schedule (Chapter 24)
   a. Establish a meaningful task set
   b. Define a task network
   c. Use scheduling tools to develop a timeline chart
   d. Define schedule tracking mechanisms

The availability of historical information has a strong influence on estimation risk. By looking back, we can emulate things that worked and improve areas where problems arose. When comprehensive software metrics (Chapter 22) are available for past projects, estimates can be made with greater assurance, schedules can be established to avoid past difficulties, and overall risk is reduced.

"It is the mark of an instructed mind to rest satisfied with the degree of precision that the nature of the subject admits, and not to seek exactness when only an approximation of the truth is possible."

Aristotle

Estimation risk is measured by the degree of uncertainty in the quantitative estimates established for resources, cost, and schedule. If project scope is poorly understood or project requirements are subject to change, uncertainty and estimation risk become dangerously high. The planner, and more importantly, the customer should recognize that variability in software requirements means instability in cost and schedule.

However, a project manager should not become obsessive about estimation. Modern software engineering approaches (e.g., incremental process models) take an iterative view of development. In such approaches, it is possible—although not always politically acceptable—to revisit the estimate (as more information is known) and revise it when the customer makes changes to requirements.

objective

23.2 The Project Planning Process

ADVICE

The more you know, the better you estimate. Therefore, update your estimates as the project progresses.

The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost, and schedule. In addition, estimates should attempt to define best-case and worst-case scenarios so that project outcomes can be bounded. Although there is an inherent degree of uncertainty, the software team embarks on a plan that has been established as a consequence of planning tasks. Therefore, the plan must be adapted and updated as the project proceeds. In the following sections, each of the activities associated with software project planning is discussed.

Task Set

Task Set for Project Planning

1. Establish project scope
2. Determine feasibility
3. Analyze risks (Chapter 25)
4. Define required resources
   a. Determine human resources required
   b. Define reusable software resources
   c. Identify environmental resources
5. Estimate cost and effort
   a. Decompose the problem
   b. Develop two or more estimates using size, function points, process tasks, or use-cases
   c. Reconcile the estimates
6. Develop a project schedule (Chapter 24)
   a. Establish a meaningful task set
   b. Define a task network
   c. Use scheduling tools to develop a timeline chart
   d. Define schedule tracking mechanisms

23.3 Software Scope and Feasibility

KEY POINT
Although there are many reasons for uncertainty, incomplete information about problem requirements dominates.

Software scope describes the functions and features that are to be delivered to end-users, the data that are input and output, the "content" that is presented to users as a consequence of using the software, and the performance, constraints, interfaces, and reliability that *bound* the system. Scope is defined using one of two techniques:

1. A narrative description of software scope is developed after communication with all stakeholders.
2. A set of use-cases[2] is developed by end-users.

Functions described in the statement of scope (or within the use-cases) are evaluated and in some cases refined to provide more detail prior to the beginning of estimation. Because both cost and schedule estimates are functionally oriented, some degree of decomposition is often useful. Performance considerations encompass processing and response time requirements. Constraints identify limits placed on the software by external hardware, available memory, or other existing systems.

ADVICE
Project feasibility is important, but a consideration of business need is even more important. It does no good to build a high-tech system or product that no one wants.

Once scope has been identified (with the concurrence of the customer), it is reasonable to ask: Can we build software to meet this scope? Is the project feasible? All too often, software engineers rush past these questions (or are pushed past them by impatient managers or customers), only to become mired in a project that is doomed from the onset. Putnam and Myers [PUT97a] address this issue when they write:

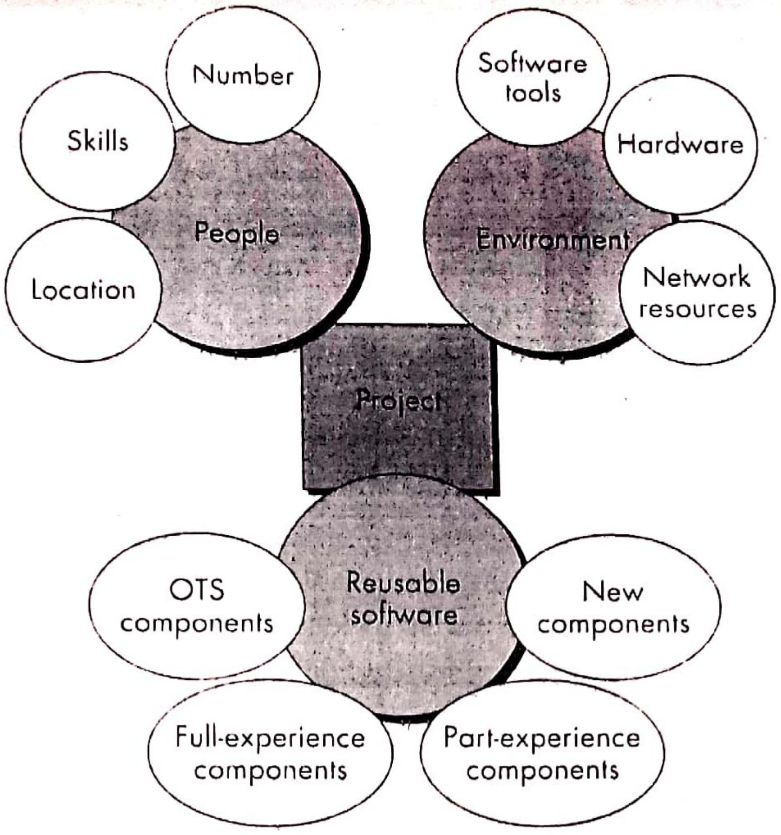
> [N]ot everything imaginable is feasible, not even in software, evanescent as it may appear to outsiders. On the contrary, software feasibility has four solid dimensions: *Technology*—Is a project technically feasible? Is it within the state of the art? Can defects be reduced to a level matching the application's needs? *Finance*—Is it financially feasible? Can development be completed at a cost the software organization, its client, or the market can afford? *Time*—Will the project's time-to-market beat the competition? *Resources*—Does the organization have the resources needed to succeed?

Putnam and Myers correctly suggest that scoping is not enough. Once scope is understood, the software team and others must work to determine if it can be done within the dimensions just noted. This is a crucial, although often overlooked, part of the estimation process.

23.4 Resources

The second planning task is estimation of the resources required to accomplish the software development effort. Figure 23.1 depicts the three major categories of software engineering resources—people, reusable software components, and the development environment (hardware and software tools). Each resource is specified with

2 Use-cases have been discussed in detail throughout Parts 2 and 3 of this book. A use-case is a scenario-based description of the user's interaction with the software from the user's point of view.



four characteristics: description of the resource; a statement of availability; time when the resource will be required; duration of time that resource will be applied. The last two characteristics can be viewed as a *time window*. Availability of the resource for a specified window must be established at the earliest practical time.

23.4.1 Human Resources

The planner begins by evaluating software scope and selecting the skills required to complete development. Both organizational position (e.g., manager, senior software engineer) and specialty (e.g., telecommunications, database, client/server) are specified. For relatively small projects (a few person-months), a single individual may perform all software engineering tasks, consulting with specialists as required. For larger projects, the software team may be geographically dispersed across a number of different locations. Hence, the location of each human resource is specified.

The number of people required for a software project can be determined only after an estimate of development effort (e.g., person-months) is made. Techniques for estimating effort are discussed later in this chapter.

23.4.2 Reusable Software Resources

Component-based software engineering (Chapter 30) emphasizes reusability—that is, the creation and reuse of software building blocks [HOO91]. Such building blocks, often called *components*, must be cataloged for easy reference, standardized for easy application, and validated for easy integration.

Bennatan [BEN92] suggests four software resource categories that should be considered as planning proceeds:

at inte-
of
nents
cant
integra-
ten
arious
e

Off-the-shelf components. Existing software can be acquired from a third party or has been developed internally for a past project. COTS (commercial off-the-shelf) components are purchased from a third party, are ready for use on the current project, and have been fully validated.

Full-experience components. Existing specifications, designs, code, or test data developed for past projects are similar to the software to be built for the current project. Members of the current software team have had full experience in the application area represented by these components. Therefore, modifications required for full-experience components will be relatively low-risk.

Partial-experience components. Existing specifications, designs, code, or test data developed for past projects are related to the software to be built for the current project but will require substantial modification. Members of the current software team have only limited experience in the application area represented by these components. Therefore, modifications required for partial-experience components have a fair degree of risk.

New components. Software components must be built by the software team specifically for the needs of the current project.

Ironically, reusable software components are often neglected during planning, only to become a paramount concern during the development phase of the software process. It is better to specify software resource requirements early. In this way technical evaluation of the alternatives can be conducted and timely acquisition can occur.

23.4.3 Environmental Resources

The environment that supports a software project, often called the *software engineering environment* (SEE), incorporates hardware and software. Hardware provides a platform that supports the tools (software) required to produce the work products that are an outcome of good software engineering practice.[3] Because most software organizations have multiple constituencies that require access to the SEE, a project planner must prescribe the time window required for hardware and software and verify that these resources will be available.

When a computer-based system (incorporating specialized hardware and software) is to be engineered, the software team may require access to hardware elements being developed by other engineering teams. For example, software for a numerical control (NC) used on a class of machine tools may require a specific machine tool (e.g., an

3 Other hardware—the target environment—is the computer(s) on which the software will execute when it has been released to the end-user.

Bennatan [BEN92] suggests four software resource categories that should be considered as planning proceeds:

CE

hat inte-
ety of
ionents
ficant
integra-
ften
arious
e

Off the shelf components. Existing software can be acquired from a third party or has been developed internally for a past project. COTS (commercial off-the-shelf) components are purchased from a third party, are ready for use on the current project, and have been fully validated.

Full experience components. Existing specifications, designs, code, or test data developed for past projects are similar to the software to be built for the current project. Members of the current software team have had full experience in the application area represented by these components. Therefore, modifications required for full-experience components will be relatively low-risk.

Partial experience components. Existing specifications, designs, code, or test data developed for past projects are related to the software to be built for the current project but will require substantial modification. Members of the current software team have only limited experience in the application area represented by these components. Therefore, modifications required for partial-experience components have a fair degree of risk.

New components. Software components must be built by the software team specifically for the needs of the current project.

Ironically, reusable software components are often neglected during planning, only to become a paramount concern during the development phase of the software process. It is better to specify software resource requirements early. In this way technical evaluation of the alternatives can be conducted and timely acquisition can occur.

23.4.3 Environmental Resources

The environment that supports a software project, often called the *software engineering environment* (SEE), incorporates hardware and software. Hardware provides a platform that supports the tools (software) required to produce the work products that are an outcome of good software engineering practice.[3] Because most software organizations have multiple constituencies that require access to the SEE, a project planner must prescribe the time window required for hardware and software and verify that these resources will be available.

When a computer-based system (incorporating specialized hardware and software) is to be engineered, the software team may require access to hardware elements being developed by other engineering teams. For example, software for a numerical control (NC) used on a class of machine tools may require a specific machine tool (e.g., an

3 Other hardware—the target environment—is the computer(s) on which the software will execute when it has been released to the end-user.

NC lathe) as part of the validation test step; a software project for advanced page-layout may need a high-quality printer at some point during development. Each hardware element must be specified by the software project planner.

23.5 Software Project Estimation

Software is the most expensive element of virtually all computer-based systems. For complex, custom systems, a large cost estimation error can make the difference between profit and loss. Cost overrun can be disastrous for the developer.

"In an age of outsourcing and increased competition, the ability to estimate more accurately . . . has emerged as a critical success factor for many IT groups."

Rob Thomsett

:E vare fort is a nent of 's emember 's (e.g., environ- s, travel, space, st also

Software cost and effort estimation will never be an exact science.[4] Too many variables—human, technical, environmental, political—can affect the ultimate cost of software and effort applied to develop it. However, software project estimation can be transformed from a black art to a series of systematic steps that provide estimates with acceptable risk. To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project (obviously, we can achieve 100 percent accurate estimates after the project is complete!).
2. Base estimates on similar projects that have already been completed.
3. Use relatively simple decomposition techniques to generate project cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation.

Unfortunately, the first option, however attractive, is not practical. Cost estimates must be provided "up front." However, we should recognize that the longer we wait, the more we know, and the more we know, the less likely we are to make serious errors in our estimates.

The second option can work reasonably well, if the current project is quite similar to past efforts and other project influences (e.g., the software team, the customer, business conditions, the SEE, deadlines) are roughly equivalent. Unfortunately, past experience has not always been a good indicator of future results.

The remaining options are viable approaches to software project estimation. Ideally, the techniques noted for each option should be applied in tandem; each used as a cross-check for the other. Decomposition techniques take a "divide and conquer" approach to software project estimation. By decomposing a project into major func-

4 Bennatan [BEN03] reports that 40 percent of software developers continue to struggle with estimation and that software size and development time are very difficult to estimate accurately.

tions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion. Empirical estimation models can be used to complement decomposition techniques and offer a potentially valuable estimation approach in their own right. These models are discussed in Section 23.7.

Each of the viable software cost estimation options is only as good as the historical data used to seed the estimate. If no historical data exist, costing rests on a very shaky foundation. In Chapter 22, we examined the characteristics of some of the software metrics that provide the basis for historical estimation data.

23.6 Decomposition Techniques

Software project estimation is a form of problem solving, and in most cases, the problem to be solved (i.e., developing a cost and effort estimate for a software project) is too complex to be considered in one piece. For this reason, we decompose the problem, recharacterizing it as a set of smaller (and hopefully, more manageable) problems.

In Chapter 21, the decomposition approach was discussed from two different points of view: decomposition of the problem and decomposition of the process. Estimation uses one or both forms of partitioning. But before an estimate can be made, the project planner must understand the scope of the software to be built and generate an estimate of its "size."

23.6.1 Software Sizing

ɔINT

ɜ" of software
lt can be
d using a direct
, LOC, or an
neasure, FP.

The accuracy of a software project estimate is predicated on a number of things: (1) the degree to which the planner has properly estimated the size of the product to be built; (2) the ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects); (3) the degree to which the project plan reflects the abilities of the software team; and (4) the stability of product requirements and the environment that supports the software engineering effort.

In this section, we consider the *software sizing* problem. Because a project estimate is only as good as the estimate of the size of the work to be accomplished, sizing represents the project planner's first major challenge. In the context of project planning, *size* refers to a quantifiable outcome of the software project. If a direct approach is taken, size can be measured in lines of code (LOC). If an indirect approach is chosen, size is represented as function points (FP).

Putnam and Myers [PUT92] suggest four different approaches to the sizing problem:

łow do
ve size the
are that
planning
ld?

- *"Fuzzy logic" sizing.* To apply this approach, the planner must identify the type of application, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range.
- *Function point sizing.* The planner develops estimates of the information domain characteristics discussed in Chapter 15.

Scanned by CamScanner

- *Standard component sizing*. Software is composed of a number of different "standard components" that are generic to a particular application area. For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions. The project planner estimates the number of occurrences of each standard component and then uses historical project data to determine the delivered size per standard component.
- *Change sizing*. This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project. The planner estimates the number and type (e.g., reuse, adding code, changing code, deleting code) of modifications that must be accomplished.

Putnam and Myers suggest that the results of each of these sizing approaches be combined statistically to create a *three-point* or *expected value* estimate. This is accomplished by developing optimistic (low), most likely, and pessimistic (high) values for size and combining them using Equation (23-1) described in the next section.

23.6.2 Problem-Based Estimation

In Chapter 22, lines of code and function points were described as measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

FP-
on
n?

LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the planner may choose another component for sizing such as classes or objects, changes, or business processes affected.

Baseline productivity metrics (e.g., LOC/pm or FP/pm[5]) are then applied to the appropriate estimation variable, and cost or effort for the function is derived. Function estimates are combined to produce an overall estimate for the entire project.

'or

It is important to note, however, that there is often substantial scatter in productivity metrics for an organization, making the use of a single baseline productivity metric suspect. In general, LOC/pm or FP/pm averages should be computed by project domain. That is, projects should be grouped by team size, application area, complexity, and other relevant parameters. Local domain averages should then be computed. When a new project is estimated, it should first be allocated to a domain,

5 The acronym *pm* means person-month of effort.

- *Standard component sizing.* Software is composed of a number of different "standard components" that are generic to a particular application area. For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions. The project planner estimates the number of occurrences of each standard component and then uses historical project data to determine the delivered size per standard component.
- *Change sizing.* This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project. The planner estimates the number and type (e.g., reuse, adding code, changing code, deleting code) of modifications that must be accomplished.

Putnam and Myers suggest that the results of each of these sizing approaches be combined statistically to create a *three point* or *expected value* estimate. This is accomplished by developing optimistic (low), most likely, and pessimistic (high) values for size and combining them using Equation (23-1) described in the next section.

23.6.2 Problem-Based Estimation

In Chapter 22, lines of code and function points were described as measures from which productivity metrics can be computed. LOC and FP data are used in two ways during software project estimation: (1) as an estimation variable to "size" each element of the software and (2) as baseline metrics collected from past projects and used in conjunction with estimation variables to develop cost and effort projections.

lo
nd FP-
ation
non?

LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. The project planner begins with a bounded statement of software scope and from this statement attempts to decompose software into problem functions that can each be estimated individually. LOC or FP (the estimation variable) is then estimated for each function. Alternatively, the planner may choose another component for sizing such as classes or objects, changes, or business processes affected.

cs for
o
ny
his

Baseline productivity metrics (e.g., LOC/pm or FP/pm[5]) are then applied to the appropriate estimation variable, and cost or effort for the function is derived. Function estimates are combined to produce an overall estimate for the entire project.

It is important to note, however, that there is often substantial scatter in productivity metrics for an organization, making the use of a single baseline productivity metric suspect. In general, LOC/pm or FP/pm averages should be computed by project domain. That is, projects should be grouped by team size, application area, complexity, and other relevant parameters. Local domain averages should then be computed. When a new project is estimated, it should first be allocated to a domain,

5 The acronym *pm* means person-month of effort.

and then the appropriate domain average for productivity should be used in generating the estimate.

The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of the partitioning. When LOC is used as the estimation variable, decomposition is absolutely essential and is often taken to considerable levels of detail. The greater the degree of partitioning, the more likely reasonably accurate estimates of LOC can be developed.

EY POINT

FP estimates, mposition focuses nformation domain cteristics

For FP estimates, decomposition works differently. Rather than focusing on function, each of the five information domain characteristics as well as the 14 complexity adjustment values discussed in Chapter 15 are estimated. The resultant estimates can then be used to derive a FP value that can be tied to past data and used to generate an estimate.

Regardless of the estimation variable that is used, the project planner begins by estimating a range of values for each function or information domain value. Using historical data or (when all else fails) intuition, the planner estimates an optimistic, most likely, and pessimistic size value for each function or count for each information domain value. An implicit indication of the degree of uncertainty is provided when a range of values is specified.

"How do we compute the pected value" software ?

A three-point or expected-value can then be computed. The *expected value* for the estimation variable (size), S, can be computed as a weighted average of the optimistic (s_{opt}), most likely (s_m), and pessimistic (s_{pess}) estimates. For example,

$$S = (s_{opt} + 4s_m + s_{pess})/6 \tag{23-1}$$

gives heaviest credence to the "most likely" estimate and follows a beta probability distribution. We assume that there is a very small probability the actual size result will fall outside the optimistic or pessimistic values.

Once the expected value for the estimation variable has been determined, historical LOC or FP productivity data are applied. Are the estimates correct? The only reasonable answer to this question is: We can't be sure. Any estimation technique, no matter how sophisticated, must be cross-checked with another approach. Even then, common sense and experience must prevail.

23.6.3 An Example of LOC-Based Estimation

As an example of LOC and FP problem-based estimation techniques, let us consider a software package to be developed for a computer-aided design application for mechanical components. The software is to execute on an engineering workstation and must interface with various peripherals including a mouse, digitizer, high-resolution color display, and laser printer. A preliminary statement of software scope can be developed:

> The mechanical CAD software will accept two- and three-dimensional geometric data from an engineer. The engineer will interact and control the CAD system through a user interface that will exhibit characteristics of good human/machine interface design. All

ADVICE

modern applica- reside on a ork or are part of nt/server archi- e. Therefore, be hat your tes include the equired to "infrastruc- oftware.

geometric data and other supporting information will be maintained in a CAD database. Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter.

This statement of scope is preliminary—it is not bounded. Every sentence would have to be expanded to provide concrete detail and quantitative bounding. For example, before estimation can begin, the planner must determine what "characteristics of good human/machine interface design" means or what the size and sophistication of the "CAD database" are to be.

For our purposes, we assume that further refinement has occurred and that the major software functions listed in Figure 23.2 are identified. Following the decomposition technique for LOC, an estimation table, shown in Figure 23.2, is developed. A range of LOC estimates is developed for each function. For example, the range of LOC estimates for the 3D geometric analysis function is optimistic—4600 LOC, most likely—6900 LOC, and pessimistic—8600 LOC. Applying Equation (23-1), the expected value for the 3D geometric analysis function is 6800 LOC. Other estimates are derived in a similar fashion. By summing vertically in the estimated LOC column, an estimate of 33,200 lines of code is established for the CAD system.

VICE

ccumb to the n to use this our project You should ther result ferent

A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC/pm. Based on a burdened labor rate of $8,000 per month, the cost per line of code is approximately $13. Based on the LOC estimate and the historical productivity data, the total estimated project cost is $431,000 and the estimated effort is 54 person-months.[6]

23.2

on the hods

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	33,200

6 Estimates are rounded to the nearest $1,000 and person month. Further precision is unnecessary and unrealistic, given the limitation of estimation accuracy.

ICE

'ern applica-
on a
are part of
ver archi-
refore, be
ur
clude the
ed to
rastruc-
re.

> geometric data and other supporting information will be maintained in a CAD database. Design analysis modules will be developed to produce the required output, which will be displayed on a variety of graphics devices. The software will be designed to control and interact with peripheral devices that include a mouse, digitizer, laser printer, and plotter.

This statement of scope is preliminary—it is not bounded. Every sentence would have to be expanded to provide concrete detail and quantitative bounding. For example, before estimation can begin, the planner must determine what "characteristics of good human/machine interface design" means or what the size and sophistication of the "CAD database" are to be.

For our purposes, we assume that further refinement has occurred and that the major software functions listed in Figure 23.2 are identified. Following the decomposition technique for LOC, an estimation table, shown in Figure 23.2, is developed. A range of LOC estimates is developed for each function. For example, the range of LOC estimates for the 3D geometric analysis function is optimistic—4600 LOC, most likely—6900 LOC, and pessimistic—8600 LOC. Applying Equation (23-1), the expected value for the 3D geometric analysis function is 6800 LOC. Other estimates are derived in a similar fashion. By summing vertically in the estimated LOC column, an estimate of 33,200 lines of code is established for the CAD system.

E

ib to the
use this
project
should
result
t

A review of historical data indicates that the organizational average productivity for systems of this type is 620 LOC/pm. Based on a burdened labor rate of $8,000 per month, the cost per line of code is approximately $13. Based on the LOC estimate and the historical productivity data, the total estimated project cost is $431,000 and the estimated effort is 54 person-months.[6]

.2

e
ds

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	33,200

6 Estimates are rounded to the nearest $1,000 and person month. Further precision is unnecessary and unrealistic, given the limitation of estimation accuracy.

SAFEHOME

Estimating

The scene: Doug Miller's office as project planning begins.

The players: Doug Miller (manager of the *SafeHome* software engineering team) and Vinod Raman, Jamie Lazar, and other members of the product software engineering team.

The conversation:

Doug: We need to develop an effort estimate for the project, and then we've got to define a micro-schedule for the first increment and a macro schedule for the remaining increments.

Vinod (nodding): Okay, but we haven't defined any increments yet.

Doug: True, but that's why we need to estimate.

Jamie (frowning): You want to know how long it's going to take us?

Doug: Here's what I need. First, we need to functionally decompose the *SafeHome* software . . . at a high level . . . then we've got to estimate the number of lines of code that each function will take . . . then. . . .

Jamie: Whoa! How are we supposed to do that?

Vinod: I've done it on past projects. You use use-cases, determine the functionality required to implement each, guesstimate the LOC count for each piece of the function. The best approach is to have everyone do it independently and then compare results.

Doug: Or you can do a functional decomposition for the entire project.

Jamie: But that'll take forever, and we've got to get started.

Vinod: No . . . it can be done in a few hours . . . this morning, in fact.

Doug: I agree . . . we can't expect exactitude, just a ball-park idea of what the size of *SafeHome* will be.

Jamie: I think we should just estimate effort . . . that's all.

Doug: We'll do that too. Then use both estimates as a cross check.

Vinod: Let's go do it. . . .

23.6.4 An Example of FP-Based Estimation

Decomposition for FP-based estimation focuses on information domain values rather than software functions. Referring to the table presented in Figure 23.3, the project planner estimates external inputs, external outputs, external inquiries, internal logical files, and external interface files for the CAD software. FP are computed using the technique discussed in Chapter 15. For the purposes of this estimate, the

FIGURE 23.3 Estimating information domain values

Information domain value	Opt.	Likely	Pess.	Est. count	Weight	FP count
Number of external inputs	20	24	30	24	4	97
Number of external outputs	12	15	22	16	5	78
Number of external inquiries	16	22	28	22	5	88
Number of internal logical files	4	4	5	4	10	42
Number of external interface files	2	2	3	2	7	15
Count total						320

Factor	Value
1. Backup and recovery	4
2. Data communications	2
3. Distributed processing	0
4. Performance critical	4
5. Existing operating environment	3
6. On-line data entry	4
7. Input transaction over multiple screens	5
8. Files updated online	3
9. Information domain values complex	5
10. Internal processing complex	5
11. Code designed for reuse	4
12. Conversion/installation in design	3
13. Multiple installations	5
14. Application designed for change	5
Value adjustment factor	**1.17**

Finally, the estimated number of FP is derived:

$$FP_{estimated} = \text{count-total} \times [0.65 + 0.01 \times \Sigma (F_i)]$$

$$FP_{estimated} = 375$$

The organizational average productivity for systems of this type is 6.5 FP/pm. Based on a burdened labor rate of $8,000 per month, the cost per FP is approximately $1,230. Based on the FP estimate and the historical productivity data, the total estimated project cost is $461,000 and the estimated effort is 58 person-months.

23.6.5 Process-Based Estimation

The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.

Like problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of framework activities must be performed for each function. Functions and related framework activities[7] may be represented as part of a table similar to the one presented in Figure 23.4.

7 The framework activities chosen for this project differ somewhat from the generic activities discussed in Chapter 2. They are customer communication (CC), planning, risk analysis, engineering, and construction/release.

Factor	Value
1 Backup and recovery	4
2 Data communications	2
3. Distributed processing	0
4. Performance critical	4
5. Existing operating environment	3
6 On-line data entry	4
7 Input transaction over multiple screens	5
8 ILFs updated online	3
9 Information domain values complex	5
10. Internal processing complex	5
11 Code designed for reuse	4
12 Conversion/installation in design	3
13 Multiple installations	5
14 Application designed for change	5
Value adjustment factor	**1.17**

Finally, the estimated number of FP is derived:

$$FP_{\text{estimated}} = \text{count-total} \times [0.65 + 0.01 \times \Sigma\,(F_i)]$$
$$FP_{\text{estimated}} = 375$$

The organizational average productivity for systems of this type is 6.5 FP/pm. Based on a burdened labor rate of $8,000 per month, the cost per FP is approximately $1,230. Based on the FP estimate and the historical productivity data, the total estimated project cost is $461,000 and the estimated effort is 58 person-months.

23.6.5 Process-Based Estimation

use when in

The most common technique for estimating a project is to base the estimate on the process that will be used. That is, the process is decomposed into a relatively small set of tasks and the effort required to accomplish each task is estimated.

major e

Like problem-based techniques, process-based estimation begins with a delineation of software functions obtained from the project scope. A series of framework activities must be performed for each function. Functions and related framework activities[7] may be represented as part of a table similar to the one presented in Figure 23.4.

7 The framework activities chosen for this project differ somewhat from the generic activities discussed in Chapter 2. They are customer communication (CC), planning, risk analysis, engineering, and construction/release.

Activity →	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
Task →				Analysis	Design	Code	Test		
Function ↓									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table in Figure 23.4. Average labor rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labor rate will vary for each task. Senior staff are heavily involved in early framework activities and are generally more expensive than junior staff involved in construction and release.

Costs and effort for each function and framework activity are computed as the last step. If process-based estimation is performed independently of LOC or FP estimation, we now have two or three estimates for cost and effort that may be compared and reconciled. If both sets of estimates show reasonable agreement, there is good reason to believe that the estimates are reliable. If, on the other hand, the results of these decomposition techniques show little agreement, further investigation and analysis must be conducted.

"It's best to understand the background of an estimate before you use it."

Barry Boehm and Richard Fairley

23.6.6 An Example of Process-Based Estimation

To illustrate the use of process-based estimation, we again consider the CAD software introduced in Section 23.6.3. The system configuration and all software functions remain unchanged and are indicated by project scope.

Referring to the completed process-based table shown in Figure 23.4, estimates of effort (in person-months) for each software engineering activity are provided for each CAD software function (abbreviated for brevity). The engineering and construction release activities are subdivided into the major software engineering tasks

Activity → / Task → / Function ↓	CC	Planning	Risk analysis	Engineering		Construction release		CE	Totals
				Analysis	Design	Code	Test		
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DBM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication   CE = customer evaluation

Once problem functions and process activities are melded, the planner estimates the effort (e.g., person-months) that will be required to accomplish each software process activity for each software function. These data constitute the central matrix of the table in Figure 23.4. Average labor rates (i.e., cost/unit effort) are then applied to the effort estimated for each process activity. It is very likely the labor rate will vary for each task. Senior staff are heavily involved in early framework activities and are generally more expensive than junior staff involved in construction and release.

Costs and effort for each function and framework activity are computed as the last step. If process-based estimation is performed independently of LOC or FP estimation, we now have two or three estimates for cost and effort that may be compared and reconciled. If both sets of estimates show reasonable agreement, there is good reason to believe that the estimates are reliable. If, on the other hand, the results of these decomposition techniques show little agreement, further investigation and analysis must be conducted.

"It's best to understand the background of an estimate before you use it."

Barry Boehm and Richard Fairley

23.6.6 An Example of Process-Based Estimation

To illustrate the use of process-based estimation, we again consider the CAD software introduced in Section 23.6.3. The system configuration and all software functions remain unchanged and are indicated by project scope.

Referring to the completed process-based table shown in Figure 23.4, estimates of effort (in person-months) for each software engineering activity are provided for each CAD software function (abbreviated for brevity). The engineering and construction release activities are subdivided into the major software engineering tasks shown. Gross estimates of effort are provided for customer communication, planning, and risk analysis. These are noted in the total row at the bottom of the table. Horizontal and vertical totals provide an indication of estimated effort required for analysis, design, code, and test. It should be noted that 53 percent of all effort is expended on front-end engineering tasks (requirements analysis and design), indicating the relative importance of this work.

Based on an average burdened labor rate of $8,000 per month, the total estimated project cost is $368,000, and the estimated effort is 46 person-months. If desired, labor rates could be associated with each framework activity or software engineering task and computed separately.

23.6.7 Estimation with Use-Cases

As we have noted throughout Parts 2 and 3 of this book, use-cases provide a software team with insight into software scope and requirements. However, developing an estimation approach with use-cases is problematic for the following reasons [SMI99]:

it
t
n
ing

- Use-cases are described using many different formats and styles—there is no standard form.
- Use-cases represent an external view (the user's view) of the software and are often written at different levels of abstraction.
- Use-cases do not address the complexity of the functions and features that are described.
- Use-cases do not describe complex behavior (e.g., interactions) that involves many functions and features.

Unlike a LOC or a function point, one person's "use-case" may require months of effort while another person's use-case may be implemented in a day or two.

Although a number of investigators have considered use-cases as an estimation input, no proven estimation method has emerged to date. Smith [SMI99] suggests that use-cases can be used for estimation, but only if they are considered within the context of the "structural hierarchy" that the use-cases describe.

Smith argues that any level of this structural hierarchy can be described by no more than 10 use-cases. Each of these use-cases would encompass no more than 30 distinct scenarios. Obviously, use-cases that describe a large system are written at a much higher level of abstraction (and represent considerably more development effort) than use-cases that are written to describe a single subsystem. Therefore, before use-cases can be used for estimation, the level within the structural hierarchy is established, the average length (in pages) of each use-case is determined, the type of software (e.g., real-time, business, engineering/scientific, embedded) is defined, and a rough architecture for the system is considered. Once these characteristics are established, empirical data may be used to establish the estimated number of LOC or shown. Gross estimates of effort are provided for customer communication, planning, and risk analysis. These are noted in the total row at the bottom of the table. Horizontal and vertical totals provide an indication of estimated effort required for analysis, design, code, and test. It should be noted that 53 percent of all effort is expended on front-end engineering tasks (requirements analysis and design), indicating the relative importance of this work.

Based on an average burdened labor rate of $8,000 per month, the total estimated project cost is $368,000, and the estimated effort is 46 person-months. If desired, labor rates could be associated with each framework activity or software engineering task and computed separately.

23.6.7 Estimation with Use-Cases

As we have noted throughout Parts 2 and 3 of this book, use-cases provide a software team with insight into software scope and requirements. However, developing an estimation approach with use-cases is problematic for the following reasons [SMI99]:

ıy is it
ficult
lop an
ion
ıe using
ɜs?

- Use-cases are described using many different formats and styles—there is no standard form.
- Use-cases represent an external view (the user's view) of the software and are often written at different levels of abstraction.
- Use-cases do not address the complexity of the functions and features that are described.
- Use-cases do not describe complex behavior (e.g., interactions) that involves many functions and features.

Unlike a LOC or a function point, one person's "use-case" may require months of effort while another person's use-case may be implemented in a day or two.

Although a number of investigators have considered use-cases as an estimation input, no proven estimation method has emerged to date. Smith [SMI99] suggests that use-cases can be used for estimation, but only if they are considered within the context of the "structural hierarchy" that the use-cases describe.

Smith argues that any level of this structural hierarchy can be described by no more than 10 use-cases. Each of these use-cases would encompass no more than 30 distinct scenarios. Obviously, use-cases that describe a large system are written at a much higher level of abstraction (and represent considerably more development effort) than use-cases that are written to describe a single subsystem. Therefore, before use-cases can be used for estimation, the level within the structural hierarchy is established, the average length (in pages) of each use-case is determined, the type of software (e.g., real-time, business, engineering/scientific, embedded) is defined, and a rough architecture for the system is considered. Once these characteristics are established, empirical data may be used to establish the estimated number of LOC or