MODULE 4

CHAPTER 1: PROJECT PLANNING

Project planning is one of the most important jobs of a software project manager. A manager has to break down the work into parts and assign these to project team members, anticipate problems that might arise, and prepare tentative solutions to those problems.

Project planning takes place at three stages in a project life cycle

1. **At the proposal stage:** when you are bidding for a contract to develop or provide a software system.
2. **During the project startup phase:** when you have to plan who will work on the project, how the project will be broken down into increments, how resources will be allocated across your company, etc.
3. **Periodically throughout the project**: when you modify your plan in light of experience gained and information from monitoring the progress of the work.

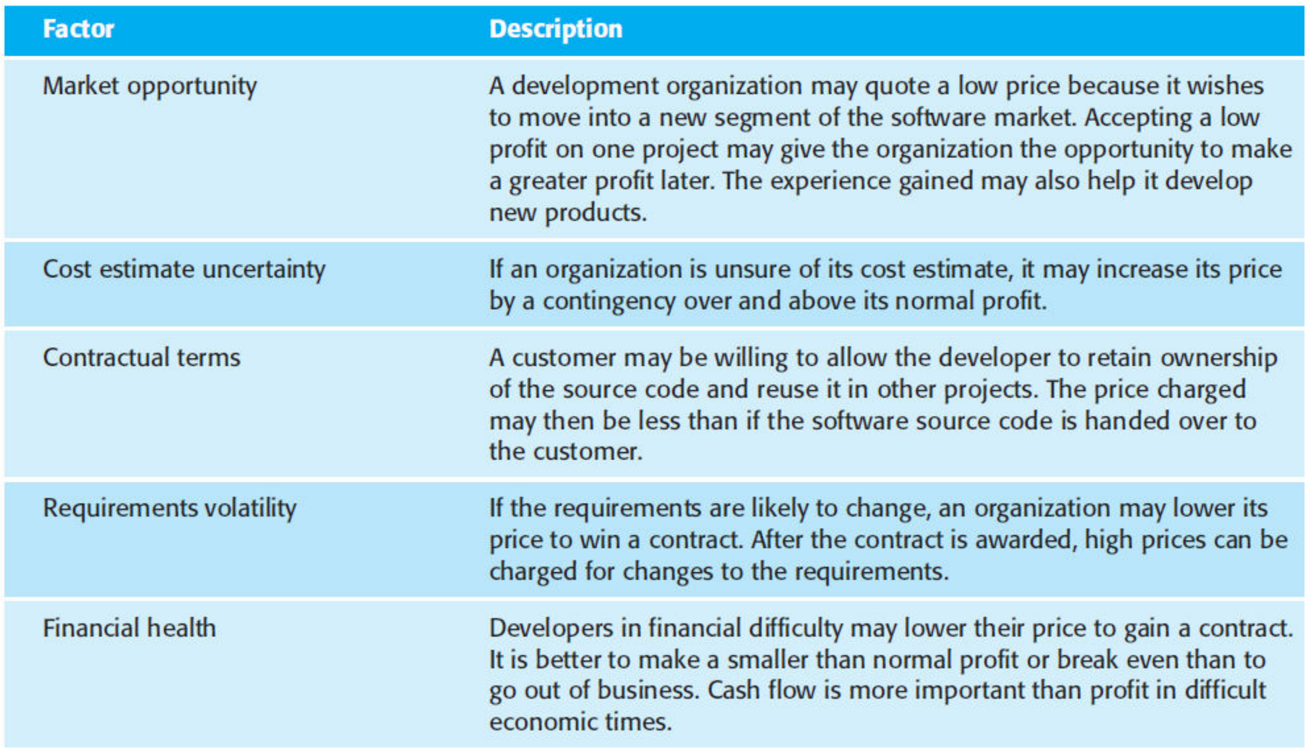
When you are bidding for a contract, you have to work out the price that you will propose to the customer for developing the software.

There are three main parameters that should use when computing the costs of a software development project:
1. Effort costs (the costs of paying software engineers and managers)
2. Hardware and software costs, including maintenance
3. Travel and training costs

1.1 Software pricing

- The price of a software product to a customer is simply the cost of development plus profit for the developer
- When calculating a price, one should take broader organizational, economic, political, and business considerations into account such as those shown in Figure 1.1
- One need to think about organizational concerns, the risks associated with the project, and the type of contract that will be used. These may cause the price to be adjusted upwards or downwards.
- As the cost of a project is only loosely related to the price quoted to a customer, 'pricing to win' is a commonly used strategy. Pricing to win means that a company has some idea of the price that the customer expects to pay and makes a bid for the contract based on the customer"s expected price.

- A project cost is agreed on the basis of an outline proposal. Negotiations then take place between client and customer to establish the detailed project specification. This specification is constrained by the agreed cost.



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

Figure 1.1: Factors affecting software pricing

1.2 Plan-Driven Development

- Plan-driven or plan-based development is an approach to software engineering where the development process is planned in detail.
- A project plan is created that records the work to be done, who will do it, the development schedule, and the work products.

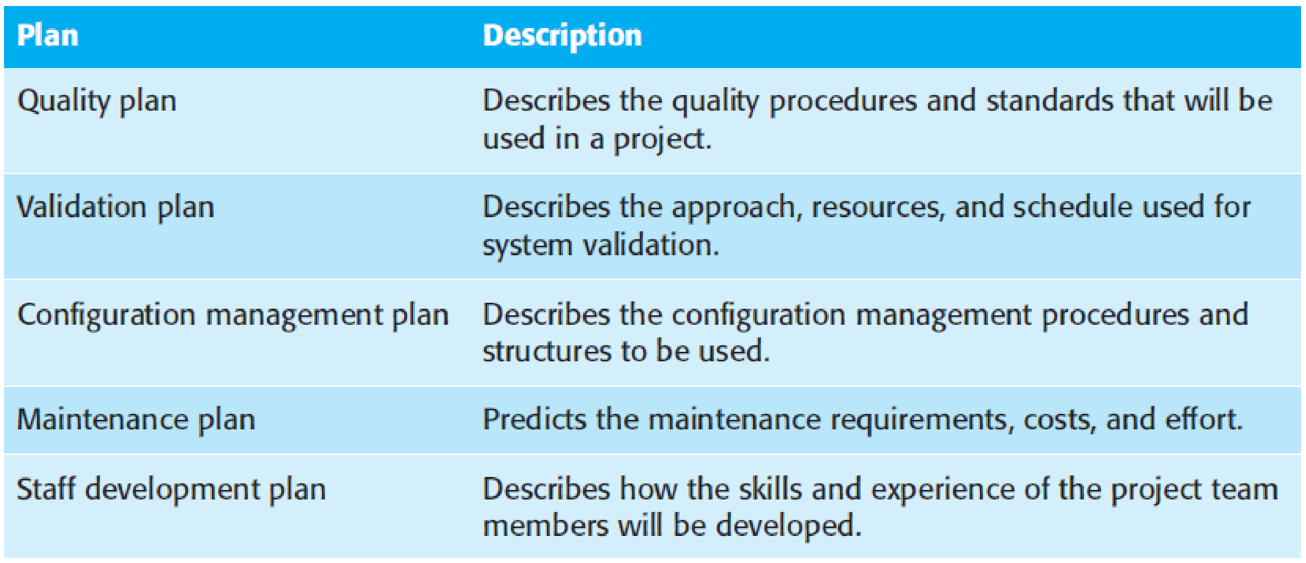
1.2.1 Project Plans

- In a plan-driven development project, a project plan sets out the resources available to the project, the work breakdown, and a schedule for carrying out the work.
- The plan should identify risks to the project and the software under development, and the approach that is taken to risk management.

Plans normally include the following sections:

1. **Introduction**: This briefly describes the objectives of the project and sets out the constraints (e.g., budget, time, etc.) that affect the management of the project.
2. **Project organization**: This describes the way in which the development team is organized, the people involved, and their roles in the team.
3. **Risk analysis:** This describes possible project risks and the risk reduction strategies that are proposed.
4. **Hardware and software resource requirements**: This specifies the hardware and support software required to carry out the development. If hardware has to be bought, estimates of the prices and the delivery schedule may be included.
5. **Work breakdown:** This sets out the breakdown of the project into activities and identifies the milestones and deliverables associated with each activity. Milestones are key stages in the project where progress can be assessed; deliverables are work products that are delivered to the customer.
6. **Project schedule**: This shows the dependencies between activities, the estimated time required to reach each milestone, and the allocation of people to activities.
7. **Monitoring and reporting mechanisms**: This defines the management reports that should be produced, when these should be produced, and the project monitoring mechanisms to be used.

A number of supplementary plans needs to develop to support other process activities such as testing and configuration management. Examples of possible supplementary plans are shown in Figure 1.2



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

Figure 1.2: Project plan supplements

1.2.2 The planning process

Project planning is an iterative process that starts when you create an initial project plan during the project startup phase.

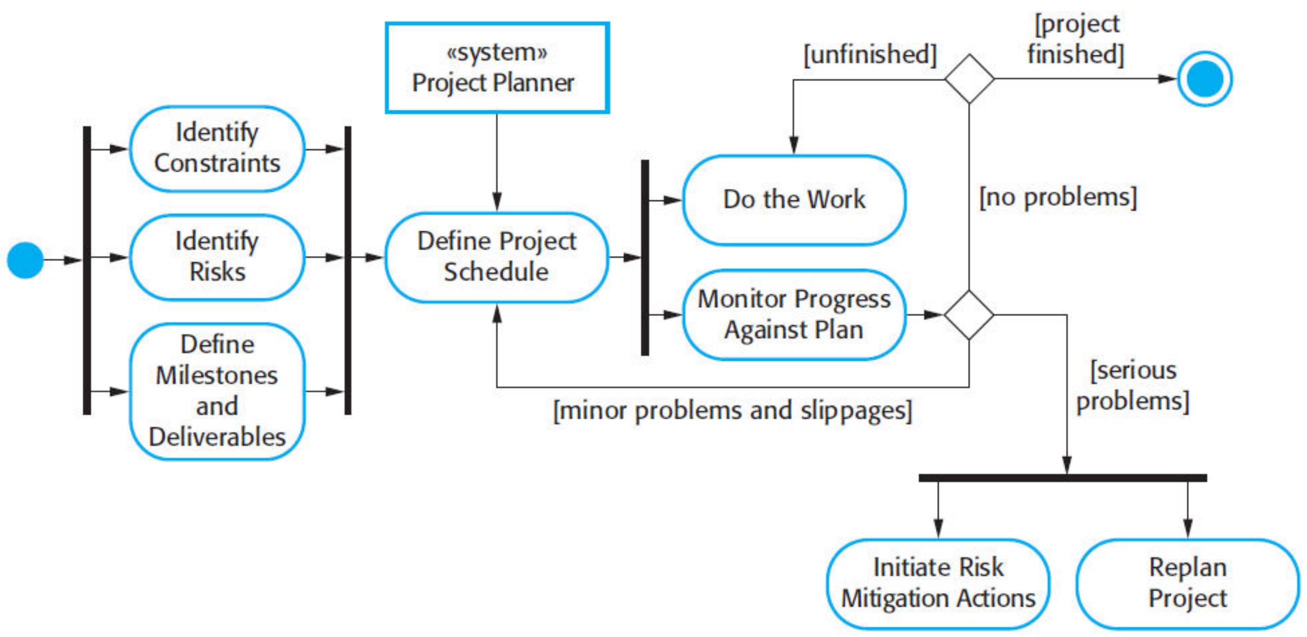


Figure 1.3: The project planning process

Figure 1.3 is a UML activity diagram that shows a typical workflow for a project planning process.

- At the beginning of a planning process, one should assess the constraints affecting the project. These constraints are the required delivery date, staff available, overall budget, available tools, and so on.
- Next identify the project milestones and deliverables. Milestones are points in the schedule against which you can assess progress. Deliverables are work products that are delivered to the customer.
- The process then enters a loop. Draw up an estimated schedule for the project and the activities defined in the schedule are initiated or given permission to continue.
- After some time (usually about two to three weeks), review the progress and note discrepancies from the planned schedule of the project.
- Because initial estimates of project parameters are inevitably approximate, minor slippages are normal and then you need to make modifications to the original plan.
- If there are serious problems with the development work that are likely to lead to significant delays, then initiate risk mitigation actions to reduce the risks of project failure.

- In conjunction with these actions, one has to replan the project. This may involve renegotiating the project constraints and deliverables with the customer. A new schedule of when work should be completed also has to be established and agreed with the customer.
- If this renegotiation is unsuccessful or the risk mitigation actions are ineffective, then arrange for a formal project technical review. The objectives of this review are to find an alternative approach that will allow the project to continue, and to check whether the project and the goals of the customer and software developer are still aligned.

1.3 Project Scheduling

- Project scheduling is the process of deciding how the work in a project will be organized as separate tasks, and when and how these tasks will be executed.
- Estimate the calendar time needed to complete each task, the effort required, and who will work on the tasks that have been identified.
- Estimate the resources needed to complete each task, such as the disk space required on a server, the time required on specialized hardware, such as a simulator, and what the travel budget will be.

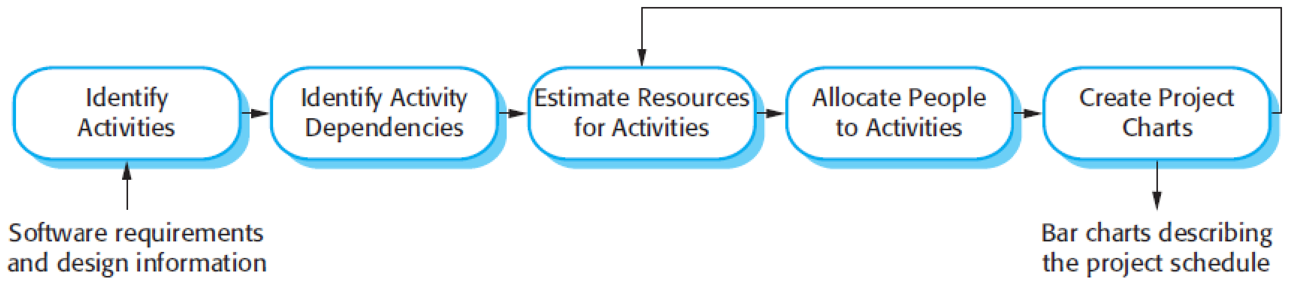

Identify Activities
Identify Activity Dependencies
Estimate Resources for Activities
Allocate People to Activities
Create Project Charts
Software requirements and design information
Bar charts describing the project schedule

Figure 1.4: The project scheduling process

Scheduling in plan-driven projects as shown in the figure 1.4 involves breaking down the total work involved in a project into separate tasks and estimating the time required to complete each task. Tasks should normally last at least 8 or 12 weeks, and no longer than 2 months.

1. From the requirements specification, identify the activities which need to be implemented.
2. Also identify the activities which are dependent on current activity.
3. An estimation of the resources that needed to be allocated to the software development need to done.
4. Distribute the activities to the staff involved in the development team.
5. The best way of representing the scheduling process is in the form of bar charts. While building this chart, you can allocate resources in parallel.

1.3.1 Schedule Representation

There are two ways of project schedule representation:

1. **Bar charts (Gantt charts):** which are calendar-based, show who is responsible for each activity, the expected elapsed time, and when the activity is scheduled to begin and end.
2. **Activity networks:** which are network diagrams, show the dependencies between the different activities making up a project

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

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

- When planning a project, also define milestones; that is, each stage in the project where a progress assessment can be made. Each milestone should be documented by a short report that summarizes the progress made and the work done.
- Milestones may be associated with a single task or with groups of related activities.
- For example, in Figure 1.5 milestone M1 is associated with task T1 and milestone M3 is associated with a pair of tasks, T2 and T4.

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

Figure 1.5: Tasks, durations, and dependencies

Building a Bar Chart

- Consider the duration column for it. In an IT industry, they will be having only 5 days as working in a week that is from Monday to Friday. Saturday and Sunday are considered as holidays.
- Task T1 needs 10 days to be completed which means it is nearly equal to 2 weeks. Task T2 needs 15 days to be completed. Task T3 starts only when Task T1 is done because T3 is dependent on T1. Here a milestone needs to be achieved (M10). When a task is dependent on more than 1 task, then take the maximum value of all the tasks.
- If the effort is less than the duration, this means that the people allocated to that task are not working full-time on it. These are called part-time tasks.
- If the effort exceeds the duration, this means that several team members are working on the task at the same time.

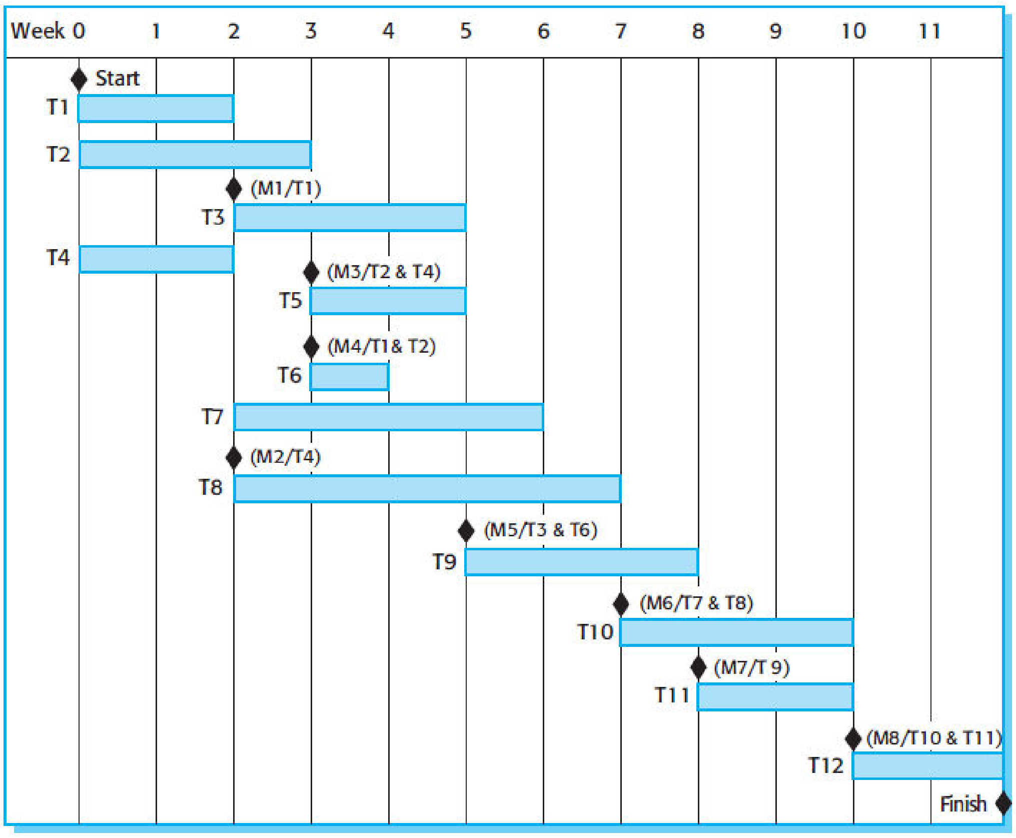


Figure 1.6: Activity bar chart

Staff Allocation Chart

People may be working on more than one task at the same time and, sometimes, they are not working on the project. Part-time assignments are shown using a diagonal line crossing the bar. The staff allocation chart is shown in figure 1.7.

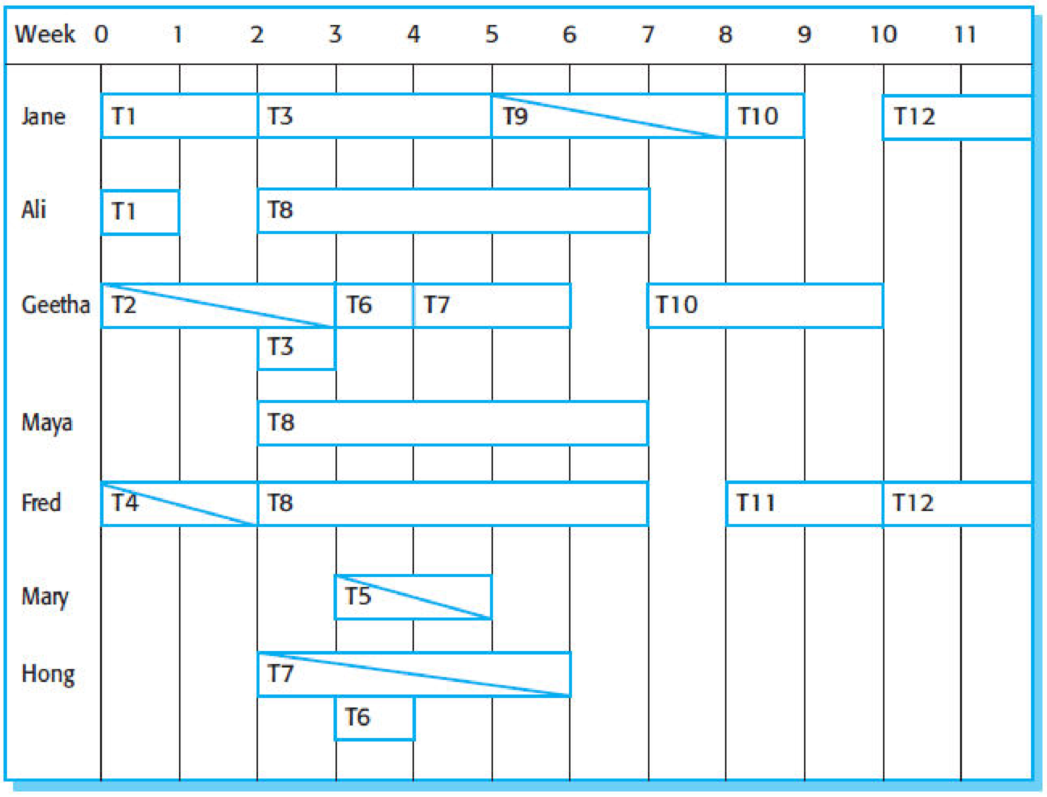


Figure 1.7 Staff allocation chart

1.4 Estimation Techniques

Project schedule estimation is difficult. The estimate is used to define the project budget and the product is adjusted so that the budget figure is realized.

<u>There are two types of techniques for estimation</u>:

<u>1. Experience-Based Techniques:</u>

- The project schedule estimation is based on the manager's experience of past projects and the application domain and the actual effort expended in these projects on activities that are related to software development
- The difficulty with experience-based techniques is that a new software project may not have much in common with previous projects. Software development changes very quickly and a project will often use unfamiliar techniques.
- If you have not worked with these techniques, your previous experience may not help to estimate the effort required, making it more difficult to produce accurate costs and schedule estimates.

<u>2. Algorithmic Cost Modeling:</u>

- In this approach, a formulaic approach is used to compute the project effort based on estimates of product attributes, such as size, and process characteristics, such as experience of staff involved.

- In both experience-based and algorithmic cost modeling, if the initial estimate of effort required is x months of effort, they found that the range may be from 0.25x to 4x of the actual effort as measured when the system was delivered.
- During development planning, estimates become more and more accurate as the project progresses as shown in figure 5.

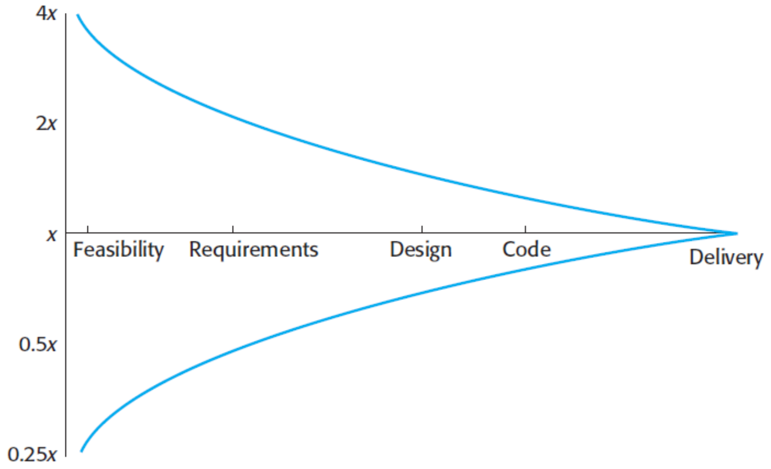


Figure 1.9: Estimate uncertainty

1.4.1 Algorithmic Cost Modeling

- Algorithmic cost modeling uses a mathematical formula to predict project costs based on estimates of the project size; the type of software being developed; and other team, process, and product factors.
- Algorithmic models for estimating effort in a software project are mostly based on a simple formula:

$$\text{Effort} = A \times \text{Size}^{B} \times M$$

  - **A** is a constant factor which depends on local organizational practices and the type of software that is developed.
  - **Size** may be either an assessment of the code size of the software or a functionality estimate expressed in function or application points.
  - The value of exponent **B** usually lies between 1 and 1.5.
  - **M** is a multiplier made by combining process, product, and development attributes

The number of lines of source code (SLOC) in the delivered system is the fundamental size metric that is used in many algorithmic cost models

1.4.2 The COCOMO II Model

- This is an empirical model that was derived by collecting data from a large number of software projects. These data were analyzed to discover the formulae that were the best fit to the observations.
- These formulae linked the size of the system and product, project and team factors to the effort to develop the system
- The sub models that are part of the COCOMO II model are shown below in the Figure 1.10

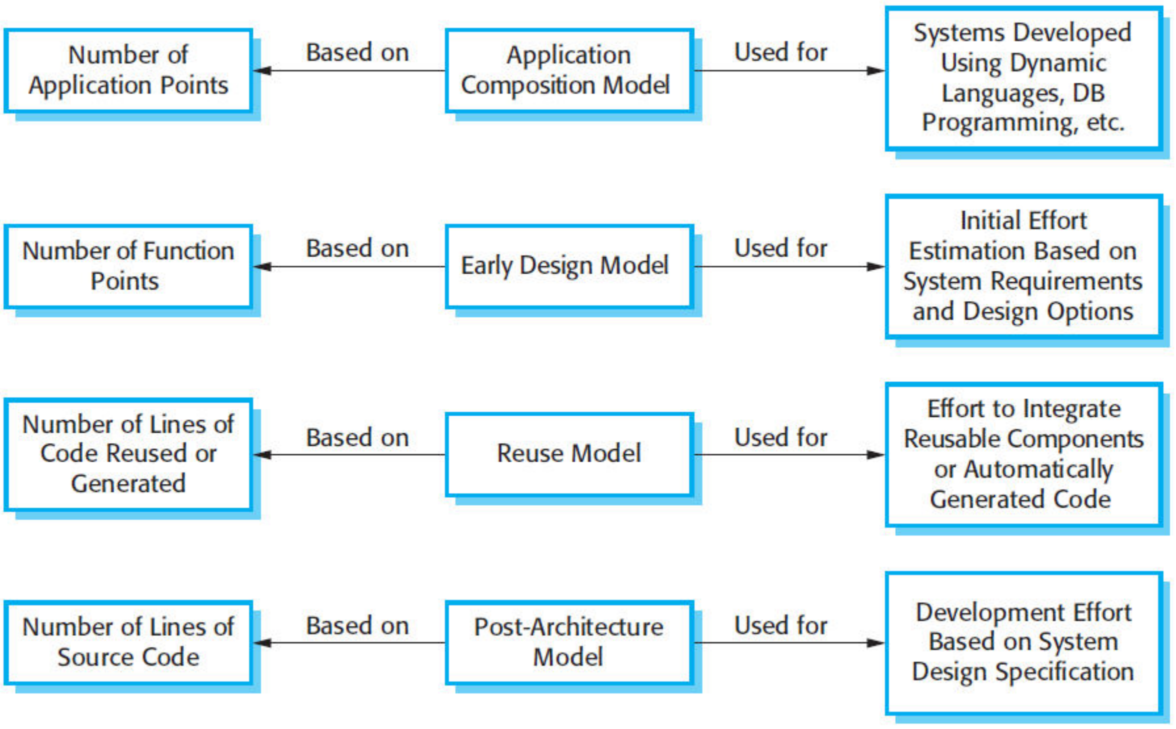

Number of Application Points
Based on
Application Composition Model
Used for
Systems Developed Using Dynamic Languages, DB Programming, etc.
Number of Function Points
Based on
Early Design Model
Used for
Initial Effort Estimation Based on System Requirements and Design Options
Number of Lines of Code Reused or Generated
Based on
Reuse Model
Used for
Effort to Integrate Reusable Components or Automatically Generated Code
Number of Lines of Source Code
Based on
Post-Architecture Model
Used for
Development Effort Based on System Design Specification

Figure 1.10: COCOMO Estimation Models

<u>1. The Application-Composition Model</u>

- This models the effort required to develop systems that are created from reusable components, scripting, or database programming.
- Software size estimates are based on application points, and a simple size/productivity formula is used to estimate the effort required.
- The number of application points in a program is a weighted estimate of the number of separate screens that are displayed, the number of reports that are produced, the number of modules in imperative programming languages, and the number of lines of scripting language or database programming code.
- Application composition involves significant software reuse.

The final formula for effort computation for system prototypes is:

$$PM = (NAP \times (1 - \%reuse/100))/PROD$$

- **PM** is the effort estimate in person-months.
- **NAP** is the total number of application points in the delivered system.
- **%reuse** is an estimate of the amount of reused code in the development.
- **PROD** is the application-point productivity, as shown in Figure 1.11. The model produces an approximate estimate as it does not take into account the additional effort involved in reuse.

Developer's experience and capability	Very low	Low	Nominal	High	Very high
ICASE maturity and capability	Very low	Low	Nominal	High	Very high
PROD (NAP/month)	4	7	13	25	50

Figure 1.11 Application point productivity

2. The early design model

- This model may be used during the early stages of a project, before a detailed architectural design for the system is available.
- Estimates are based on function points, which are then converted to number of lines of source code.
- Compute the total number of function points in a program by measuring or estimating the number of external inputs and outputs, user interactions, external interfaces, and files or database tables used by the system.
- The estimates produced at this stage are based on the standard formula for algorithmic models:

$$\text{Effort} = A \times \text{Size}^{B} \times M$$

  - Based on his own large data set, Boehm proposed that the coefficient **A** should be 2.94
  - The **Size** (KSLOC), which is the number of thousands of lines of source code. Calculate KSLOC by estimating the number of function points in the software.
  - The exponent B reflects the increased effort required as the size of the project increases. This can vary from 1.1 to 1.24 depending on the novelty of the project, the development flexibility.

3. The reuse model

- This model is used to compute the effort required to integrate reusable components and/or automatically generated program code.
- COCOMO II considers two types of reused code.
    1. **'Black-box'** code is code that can be reused without understanding the code or making changes to it. The development effort for black box code is taken to be zero.
    2. **'White box'** code has to be adapted to integrate it with new code or other reused components. Development effort is required for reuse because the code has to be understood and modified before it can work correctly in the system.

- The COCOMO II reuse model includes a formula to estimate the effort required to integrate this generated code:

$$PM_{Auto} = (ASLOC \times AT/100) / ATPROD$$

    - ASLOC is the total number of lines of reused code, including code that is automatically generated.
    - AT is the percentage of reused code that is automatically generated.
    - ATPROD is the productivity of engineers in integrating such code.

- The following formula is used to calculate the number of equivalent lines of source code:

$$ESLOC = ASLOC \times AAM$$

    - ESLOC is the equivalent number of lines of new source code.
    - ASLOC is the number of lines of code in the components that have to be changed.
    - AAM is an Adaptation Adjustment Multiplier

4. Post-architecture model

- It is used once an initial architectural design for the system is available so the sub system structure is known. Then make estimates of each part of the system.
- The starting point for estimates produced at the post-architecture level is the same basic formula used in the early design estimates:

$$PM = A \times Size^B \times M$$

Size: Make the estimate of the code size using three parameters:

1. An estimate of the total number of lines of new code to be developed (SLOC).
2. An estimate of the reuse costs based on an equivalent number of source lines of code (ESLOC), calculated using the reuse model.
3. An estimate of the number of lines of code that are likely to be modified because of changes to the system requirements.

The total code size KSLOC can be computed by adding the values of these 3 parameters.

The exponent term (B) in the effort computation formula is related to the levels of project complexity. As projects become more complex, the effects of increasing system size become more significant.

- The value of the exponent B is therefore based on five factors, as shown in Figure 7. These factors are rated on a six-point scale from 0 to 5, where 0 means 'extra high' and 5 means 'very low'.
- To calculate B, add the ratings, divide them by 100, and add the result to 1.01 to get the exponent that should be used.

Scale factor	Explanation
Precedentedness	Reflects the previous experience of the organization with this type of project. Very low means no previous experience; extra-high means that the organization is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process. Very low means a prescribed process is used; extra-high means that the client sets only general goals.
Architecture/risk resolution	Reflects the extent of risk analysis carried out. Very low means little analysis; extra-high means a complete and thorough risk analysis.
Team cohesion	Reflects how well the development team knows each other and work together. Very low means very difficult interactions; extra-high means an integrated and effective team with no communication problems.
Process maturity	Reflects the process maturity of the organization. The computation of this value depends on the CMM Maturity Questionnaire, but an estimate can be achieved by subtracting the CMM process maturity level from 5.

Figure 1.12 Scale factors used in the exponent computation in the post-architecture model

1.4.3 Project Duration and Staffing

The COCOMO model includes a formula to estimate the calendar time required to complete a project.

$$\text{TDEV} = 3 \times (\text{PM})^{(0.33 + 0.2*(B - 1.01)}$$

- TDEV is the nominal schedule for the project, in calendar months, ignoring any multiplier that is related to the project schedule.
- PM is the effort computed by the COCOMO model.
- B is the complexity-related exponent,

If B = 1.17 and PM = 60 then

$$\text{TDEV} = 3 \times (60)^{0.36} = 13 \text{ months}$$

- Adding more people in the project team reduces the productivity.
- As the project team increases in size, team members spend more time in communicating and defining the interfaces. So, this mainly increases the development schedule.
- A small number of people are needed at the start of a project to carry out the initial design. The team then builds up to a peak during the development and testing of the system, and then declines in size as the system is prepared for deployment.
- Project managers should therefore avoid adding too many staff to a project early in its lifetime.