CHAPTER - 2

Planning Software Project

2.1 Project Management Concepts

Building computer software is a complex undertaking, particularly if it involves many people working over a relatively long time. That's why software projects need to be managed. The management activities varies among people involved in a software project. A software engineer manages her day-to-day activities, planning, monitoring, and controlling technical tasks. Project managers plan, monitor, and control the work of a team of software engineers. Senior managers coordinate the interface between the business and software professionals.

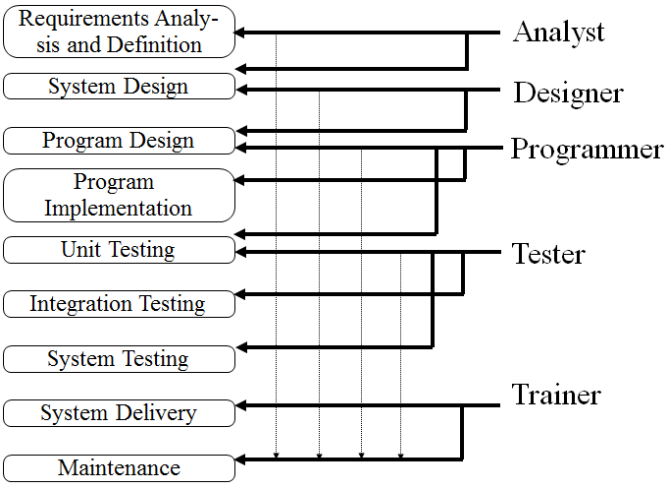
Software project management is an umbrella activity within software engineering. It begins before any technical activity is initiated and continues throughout the definition, development and lifetime support of computer software. The project management activity encompasses measurement and metrics, estimation and scheduling, risk analysis, tracking, and control. The goals of software project management are effective team, focusing their attention on customer needs and product quality.

2.2 The Management Spectrum

Effective software project management focuses on the **four P's**: people, product, process, and project. *People* must be organized to perform software work effectively. Communication with the customer and other stakeholders must occur so that *product* scope and requirements are understood. A *process* that is appropriate for the people and the product should be selected. The *project* must be planned by estimating effort and calendar time to accomplish work tasks: defining work products, establishing quality checkpoints, and identifying mechanisms to monitor and control work defined by the plan.

A. The People

The people are the primary key factor for successful organization. For the successful software production environment, any organization must perform the proper staffing, communication and coordination, work environment, performance management, training, compensation (or reward), competency analysis and development, career development, workgroup development, team/culture development, and others.



People Capability Maturity Model (People-CMM) illustrate that "every organization needs to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives by improving the quality on its staff. ".

B. The Product

Before a project can be planned, product objectives and scope should be established, alternative solutions should be considered, and technical and management constraints should be identified, without product information, it is impossible to define reasonable (and accurate) estimates of the cost, an effective assessment of risk, a realistic breakdown of project tasks, or a manageable project schedule that provides a meaningful indication of progress.

C. The Process

A software process provides the framework from which a comprehensive plan for software development can be established. Although, the detail operation to do for the project development may be differ from problem to problem but the general framework for software development remain same.

D. The Project

We conduct planned and controlled software projects for manage complexity of project. To avoid project failure, a software project manager and the software engineers who build the product must avoid a set of common warning signs, understand the critical success factors that lead to good project management, and develop a commonsense approach for planning, monitoring, and controlling the project.

2.3 The W^5HH Principles

On software process and projects, **Barry Boehm** states: "you need an organizing principle that scales down to provide simple plans for simple projects." Boehm suggests an approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources. He calls it the *W5HH Principle*, after a series of questions that lead to a definition of key project characteristics and the resultant project plan:

- *Why is the system being developed?* All stakeholders should assess the validity of business reasons for the software work. Does the business purpose justify the expenditure of people, time, and money?
- *What will be done?* The task set required for the project is defined.
- *When will it be done?* The team establishes a project schedule by identifying when project tasks are to be conducted and when milestones are to be reached.
- *Who is responsible for a function?* The role and responsibility of each member of the software team is defined.
- *Where are they located organizationally?* Not all roles and responsibilities reside within software practitioners. The customer, users, and other stakeholders also have responsibilities.
- *How will the job be done technically and managerially?* Once product scope is established, a management and technical strategy for the project must be defined.
- *How much of each resource is needed?* The answer to this question is derived by developing estimates based on answers to earlier questions.

Boehm's W5HH Principle is applicable regardless of the size or complexity of a software project. The questions noted provide you and your team with an excellent planning outline.

2.4 Project Planning Process

Software project management begins with a set of activities that are collectively called *project planning*. The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost, and schedule. Software project planning encompasses five major activities—estimation, scheduling, risk analysis, quality management planning, and change management planning. The schedule slippage, cost overrun, poor quality, and high maintenance costs for software may cause due to the lack of planning. Thus, planning attempt to define the best case and worst case scenario so that project outcome 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 these tasks. Therefore, the plan must be adapted and updated as the project proceeds because "*The more you know, the better you estimate. Therefore, update your estimates as the project progresses.*"

The **estimation** attempt to determine how much money, effort, resources, and time it will take to build a specific software-based system or product. Estimation begins with a description of the **scope** of the problem. The problem is then decomposed into a set of smaller problems, and each of these is estimated using historical data and experience as guides. Problem complexity and risk are considered before a final estimate is made. After completion of estimation, **project scheduling** is started that defines software engineering tasks and milestones, identifies who is responsible for conducting each task, and specifies the inter-task dependencies that may have a strong bearing on progress.

2.5 Software Scope

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:

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

In general sense, the software scope indicates the acceptance limit of software by end user or environment. The software scope must be unambiguous and understandable at the management and technical level. 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?"

2.6 Feasibility

The feasibility explains the acceptance of the software in different condition of technology, finance, time, resources, and process of operation, which all are explained below:

A. Technical Feasibility

The software product must be technically feasible by the use of present and near future hardware and techniques. Also, it must be feasible to afford the technical person for new techniques or features required on the software product.

B. Schedule (Time) Feasibility

The software product must be built in time, lunch in market in proper time stamp for competition, and financially beneficial. The overrun of software time will reduce the level of believe on customer for future task, may be loss the market price and popularity by incoming of same type of software from the competitor, and production cost may be much more.

C. Financial Feasibility

The software product must be cost effective so that its user or users can purchase it. Sometime the selling of large copy of a software product in low price will be more beneficial than that of high cost selling in the prospective of popularity and benefit.

D. Operational Feasibility

The user must know about the operation of the software product to use it in efficient way. Thus, the process of operation must be predefined in many ways such as training, meeting, presentation demo etc.

E. Resource Feasibility

The software development company must have the software and hardware resources to build the quality software product as the customer demand. Also, the product must be well operating on the end user machine after delivery.

2.7 Resources

The three major categories of software engineering resources are—people, reusable software components, and the development environment (hardware and software tools). Each resource is specified with four characteristics: description of the resource, a statement of availability, time when the resource will be required, and duration of time that the resource will be applied.

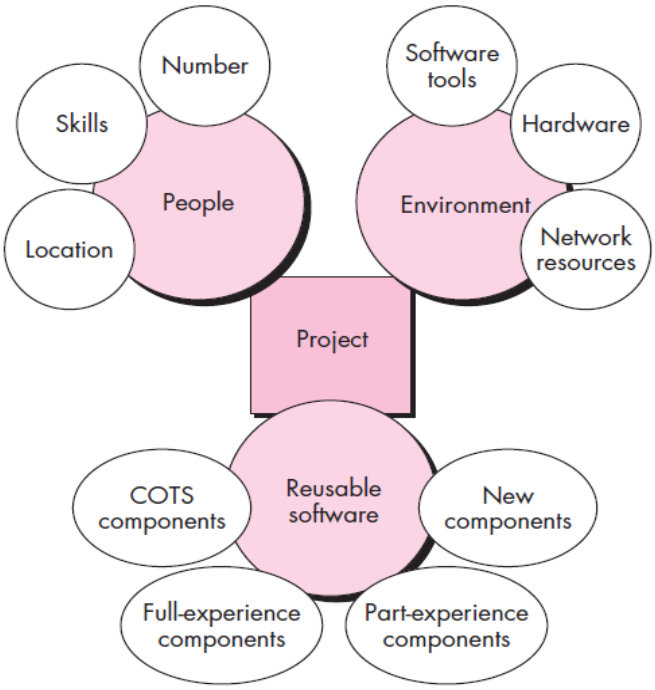


Fig. Project resources

A. Human Resources

Human are the primary resources that evaluate software scope and select the skills required to complete development. They specify both organizational position (e.g., manager, senior software engineer) and specialty (e.g., telecommunications, database, and client-server).

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. 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.

B. Reusable Software Resources

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

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

- **Off-the-shelf components:** Existing software that can be acquired from a third party or from 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 that 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 that 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.

C. 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. 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 robotic device may require a specific robot as part of the validation test step.

2.8 Software Project Estimation

Today the software is the most expensive element of all computer system and the large cost estimation error can make the difference between the profit and loss. Software cost and effort estimation will never be an exact science. 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. The cost due to late in the project (100 percent cost only after the project complete).
2. The cost of previous similar project must be analyzed.
3. Use relatively simple decomposition techniques to generate project cost and effort estimates. Such decomposition techniques may be LOC-based estimation, FP-based estimation, Process based estimation, Estimation with use case etc.
4. Use one or more empirical models (like COCOMO II) for software cost and effort estimation.

Software metrics (qualitative measures) can be categorized similarly to real world scenario. But the result from the software measurement metrics is not sufficient for the overall sizing of software because some metrics are hidden and appears at the time of software construction. The software estimation generally done in two levels: Top-down and Bottom-up approach.

Top-down estimation

Top-down estimation start at the system level and assess the overall system functionality and how this is delivered through sub-systems.

- Usable without knowledge of the system architecture and the components that might be part of the system.
- Takes into account costs such as integration, configuration management and documentation.
- Can underestimate the cost of solving difficult low-level technical problems.

Bottom-up estimation

It starts at the component level and estimate the effort required for each component. Add these efforts to reach a final estimate.

- Usable when the architecture of the system is known and components identified.
- This can be an accurate method if the system has been designed in detail.
- It may underestimate the costs of system level activities such as integration and documentation.

2.9 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 should decompose the problem, re-characterizing it as a set of smaller (and hopefully, more manageable) problems.

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."

A. Software Sizing

The accuracy of software project estimation is depends on:

- The size of product to be built.
- The human effort, calendar time, and availability of reusable software components.
- The ability and experience of the software team.
- The stability of the product requirements and the supporting environments.

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).

B. Problem-Based Estimation

In problem based estimation, productivity metrics can be computed using lines of code and function points estimation. LOC and FP estimation are distinct estimation techniques. Yet both have a number of characteristics in common. LOC and FP data are used in two ways during software project estimation: (1) 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.

The *expected value* for the estimation variable (S) can be computed as a weighted average of the optimistic (s_{opt}), most likely (s_m), and pessimistic (s_{pess}) estimates. The estimated value may be of LOC or FP estimation techniques.

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6}$$

Software measurement

The result from software measurement metric is not sufficient for the overall sizing of software because some metrics are hidden and appear at the time of software constriction. In spite of this, the measurement in the physical world can be categorized into two ways: direct measurements and indirect measurements.

- **Direct measure**: Direct measure of software process includes cost and effort applied which are easier to measure by person-month analysis. Also, it includes line of code (LOC) produced, execution speed, memory size and defects reported over some set period of time.

- **Indirect measure**: Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintenance possibilities, usability, portability, etc. They are relatively difficult to measure.

a. Line of code (LOC) measurement

LOC is simplest way to estimate the project size among all metrics available. The measure was first proposed when programs were typed on cards with one line per card. To find the LOC at the beginning of a project, divide module into sub-module and so on, until size of each module can be predicted. The project size estimation by counting the number of source instruction is an ambiguous task because there may have multiple lines of code used for commenting, header lines of codes etc. Thus, a commonly adopted convention is to count only the lines of codes that are delivered to the customers as part of the product. The LOC estimation technique has following limitations:

- The LOC will vary according to programming style i.e. complex logic may reduce some codes by the replacement of the simple logic.
- The reuse of codes will vary the program size.
- There may not be better quality on large sized programs.
- The accurate LOC are only computed after project completion.
- On project estimation, the software product not only depends on the LOC but this may vary due to analysis, design, testing etc.

For example: If

- Estimated total LOC on any project = 33,200
- Organizational average productivity = 620 LOC/pm
- Labor rate per month = $8000

Then,

- Cost per LOC = ($8000/m)/(620LOC/pm) = $13
- Total project cost = Total LOC x Cost per LOC = 33,200 x $13 = $4,31,600
- Estimated effort = Total LOC / Average productivity
  = (33,200 LOC) / (620 LOC/person month) = 54 persons

b. Function based metric (FP)

The function point (FP) metric can be used effectively as a means for measuring the functionality delivered by a system. Using historical data, the FP metric can be used to:

- Estimate the cost or effort required to design, code and test the software,
- Predicate the number of errors that will be encounter during testing,
- Forecast the number of components in implemented system.

The major limitation of FP based estimation is that the weight of items on metric is fixed which may not sufficient for all cases. Hence, this problem can be solved by providing a range of weights for each item based on simple, average or complex subjective determination.

Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)		×	3	4	6	=	
External Outputs (EOs)		×	4	5	7	=	
External Inquiries (EQs)		×	3	4	6	=	
Internal Logical Files (ILFs)		×	7	10	15	=	
External Interface Files (EIFs)		×	5	7	10	=	
Count total	→						

Fig. Computing function points

Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain (UFP) and qualitative assessments of software complexity (TCF). Information domain values are defined in the following manner:

- **Number of external inputs (EIs) (weight-4):** Each *external input* originates from a user or is transmitted from another application and provides distinct application-oriented data or control information. Inputs are often used to update *internal logical files* (ILFs). Inputs should be distinguished from inquiries, which are counted separately.
- **Number of external outputs (EOs) (weight-5):** Each *external output* is derived data within the application that provides information to the user. In this context external output refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.
- **Number of external inquiries (EQs) (weight-4):** An *external inquiry* is defined as an online input that results in the generation of some immediate software response in the form of an online output (often retrieved from an ILF).
- **Number of internal logical files (ILFs) (weight-10):** Each *internal logical file* is a logical grouping of data that resides within the application's boundary and is maintained via external inputs.
- **Number of external interface files (EIFs) (weight-10):** Each *external interface file* is a logical grouping of data that resides external to the application but provides information that may be of use to the application.

Function point (FP) = Unadjusted function point (UFP) x Technical complexity factor (TCF)

Where,

- UPF = Input x 4 + output x 5 + Inquiries x 4 + Files x 10 + Interface x 10
- TCF (varies from 0.65 to 1.35) = 0.65 + 0.01 x Degree of influence (DI)
- DI = Number of questions x weight to each question = 14 x (0 to 5) = $\sum (F_i)$]

Thus, FP = UFP x [0.65 x 0.01 x $\sum (F_i)$]

The F_i (i = 1 to 14) is *value adjustment factors* (VAF) based on responses to the following questions:

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Each of these questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential). Note that, if these all 14 technical questions have given the average value (3), then TCF = 0.65 + 0.01 x (14x3) = 1.07.

For example: If FP =UFP x TCF = 375 FP, average productivity = 6.5 FP/PM and labor rate = $8000 per month then

- Cost per FP = ($8000) / (6.5 FP/PM) = $1230
- Total project cost = 375 x $1230 = $461250
- Estimated effort = Total FP / Average productivity = 375 FP / (6.5 FP/PM) = 58 persons

C. Empirical estimation models

An estimation model reflects the population of projects from which it has been derived. Therefore, the model is domain sensitive. An estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP. Instead of using the tables described in previous sections, the resultant values for LOC or FP are plugged into the estimation model.

The empirical data that support most estimation models are derived from a limited sample of projects. For this reason, no estimation model is appropriate for all classes of software and in all development environments. Therefore, if agreement is poor, the model must be tuned and retested before it can be used.

The Structure of Estimation Models

A typical estimation model is derived using regression analysis on data collected from past software projects. The overall structure of such models takes the form: $\boldsymbol{E = A + B \times (e_v)^C}$, where A, B, and C are empirically derived constants, E is effort in person-months, and e_v is the estimation variable (either LOC or FP).

LOC-oriented estimation models	
$E = 5.2 \times (\text{KLOC})^{0.91}$	Walston-Felix model
$E = 5.5 + 0.73 \times (\text{KLOC})^{1.16}$	Bailey-Basili model
$E = 3.2 \times (\text{KLOC})^{1.05}$	Boehm simple model
$E = 5.288 \times (\text{KLOC})^{1.047}$	Doty model for KLOC > 9
FP-oriented estimation models	
$E = -91.4 + 0.355\ \text{FP}$	Albrecht and Gaffney model
$E = -37 + 0.96\ \text{FP}$	Kemerer model
$E = -12.88 + 0.405\ \text{FP}$	Small project regression model

The COCOMO II Model

The constructive cost model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm and published in 1981 on his book named Software Engineering Economics. The model uses a basic regression formula, with some parameters that are derived from historical project data and current project characteristics. With reference to the COCOMO, the COCOMO-II was developed in 1997 and finally published in 2000 in the book Software Cost Estimation with COCOMO-II.

Thus, COCOMO-II is the successor of COCOMO-81 and is better suited for estimating software development projects. It provides more support for modern software development processes and an updated project database. The need for new model came as software development technology moved from mainframe and overnight batch processing to desktop development, cost reusability and the use of off-the-self software components.
COCOMO II is actually a hierarchy of estimation models that address the following areas:

- **Application composition model:** Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
- **Early design stage model:** Used once requirements have been stabilized and basic software architecture has been established.
- **Post-architecture-stage model**: Used during the construction of the software.

Basic COCOMO-II Model

COCOMO applies to three classes of software projects:

- **Organic**: Developing well understood application programs, small experienced team
- **Semi Detached**: mix of experienced and non-experienced team, unfamiliar
- **Embedded**: strongly coupled to computer hardware

Basic COCOMO

- Effort = a (KLOC)^{b} PM
- Time = c (Effort)^{d} Months
- Number of people required = (Effort applied) / (Development time)

The program size is expressed in thousands of lines of code (KLOC). The coefficient a, b, c, and d are given as:

Software project	Effort schedule			
	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Example: The size of organic software is estimated to be 32,000 LOC. The average salary for software engineering is Rs. 15000/- per month. What will be effort and time for the completion of the project?

Solution:

- Effort applied = 2.4 x (32)$^{1.05}$ PM = 91.33 PM (Since: 32000 LOC = 32KLOC)
- Time = 2.5 x (91.33)$^{0.38}$ Month = 13.899 Months
- Cost = Time x Average salary per month = 13.899 x 15000 = Rs. 208480.85
- People required = (Effort applied) / (development time) = 6.57 = 7 persons

D. Estimation for Object-Oriented Projects

It is worthwhile to supplement conventional software cost estimation methods with a technique that has been designed explicitly for OO software. Lorenz and Kidd suggest the following approach:

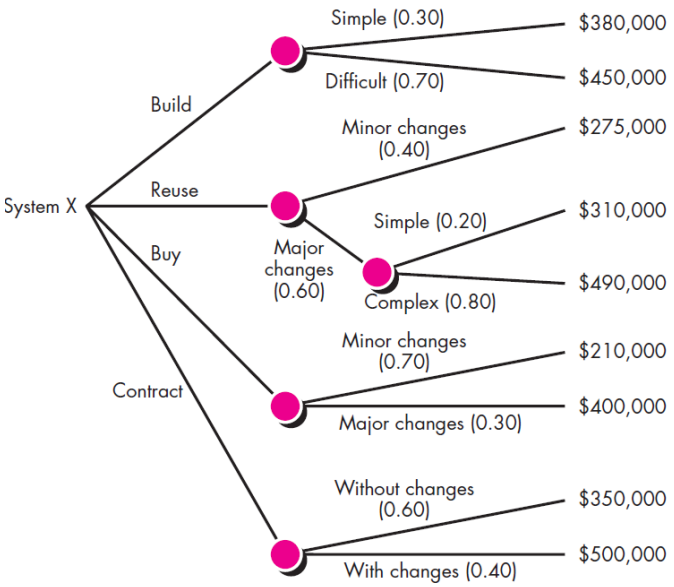
1. Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
2. Using the requirements model, develop use cases and determine a count. Recognize that the number of use cases may change as the project progresses.
3. From the requirements model, determine the number of key classes (called analysis classes).
4. Categorize the type of interface for the application and develop a multiplier for support classes: Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.

Interface Type	Multiplier
No GUI	2.0
Text-based user interface	2.25
GUI	2.5
Complex GUI	3.0

5. Multiply the total number of classes (key + support) by the average number of work units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
6. Cross-check the class-based estimate by multiplying the average number of work units per use case.

E. The make/Buy decision

In many software application areas, it is often more cost effective to acquire rather than develop computer software. Software engineering managers are faced with a make/buy decision that can be further complicated by a number of acquisition options:



- software may be purchased (or licensed) off-the-shelf,
- "full-experience" or "partial-experience" software components may be acquired and then modified and integrated to meet specific needs, or
- software may be custom built by an outside contractor to meet the purchaser's specifications

The make/buy decision is made based on the following conditions:

- Will the delivery date of the software product be sooner than that for internally developed software?
- Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?
- Will the cost of outside support (e.g., a maintenance contract) be less than the cost of internal support?

❖ Creating a Decision Tree

A decision tree is used to support the make/buy decision in similar to the given figure:

$$\text{Expected cost} = \Sigma\ (\text{path probability})_i \times (\text{estimated path cost})_i$$

where i is the decision tree path.

$$\text{Expected cost}_{build} = 0.30\ (\$380K) + 0.70\ (\$450K) = \$429K$$
$$\text{Expected cost}_{reuse} = 0.40\ (\$275K) + 0.60\ [0.20\ (\$310K) + 0.80\ (\$490K)] = \$382K$$
$$\text{Expected cost}_{buy} = 0.70\ (\$210K) + 0.30\ (\$400K) = \$267K$$
$$\text{Expected cost}_{contract} = 0.60\ (\$350K) + 0.40\ (\$500K) = \$410K$$

❖ Outsourcing

In concept, outsourcing is extremely simple. Software engineering activities are contracted to a third party who does the work at lower cost and, hopefully, higher quality. Software work conducted within a company is reduced to a contract management activity.

- On the positive side, cost savings can usually be achieved by reducing the number of software people and the facilities (e.g., computers, infrastructure) that support them.
- On the negative side, a company loses some control over the software that it needs. Since software is a technology that differentiates its systems, services, and products, a company runs the risk of putting the fate of its competitiveness into the hands of a third party.

2.10 Project Scheduling

In order to build a complex system, many software engineering tasks occur in parallel, and the result of work performed during one task may have a profound effect on work to be conducted in another task. These interdependencies are very difficult to understand without a schedule. It's also virtually impossible to assess progress on a moderate or large software project without a detailed schedule.

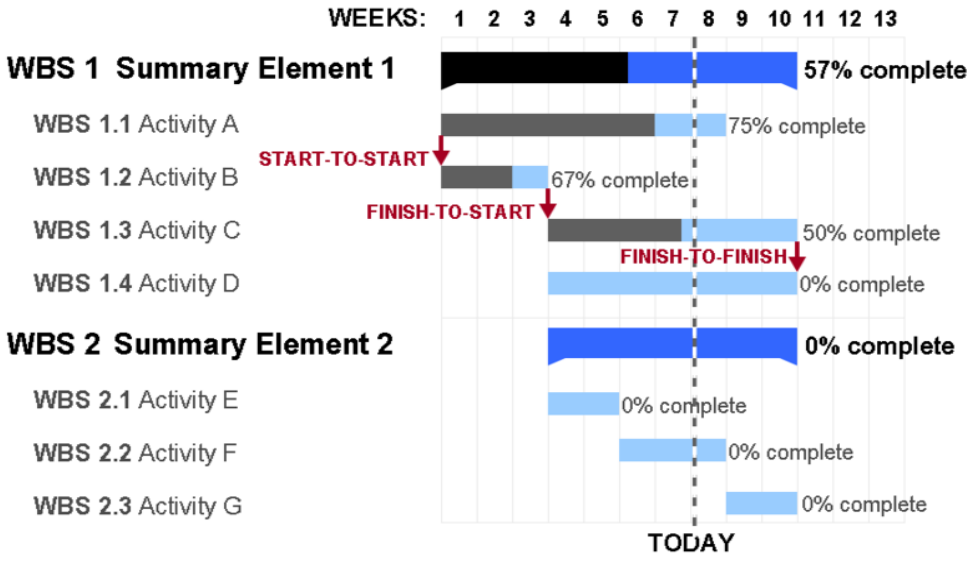
Thus, after selection of appropriate process model, identification of the software engineering tasks that have to be performed, estimation of the amount of work and the number of people, fixing the deadline, and consideration of the risks, the next process on software engineering is the project scheduling. *Software project scheduling* is an action that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks. The schedule may evolves over time according to the changes appeared on the project.

Program evaluation and review technique (PERT) and the *critical path method* (CPM) are two project scheduling methods that can be applied to software development. Both techniques are driven by information already developed in earlier project planning activities: estimates of effort, a decomposition of the product function, the selection of the appropriate process model and task set, and decomposition of the tasks that are selected. Interdependencies among tasks may be defined using a task network. Tasks, sometimes called the project *work breakdown structure* (WBS), are defined for the product as a whole or for individual functions. Both PERT and CPM provide quantitative tools that allow us to

1. determine the critical path—the chain of tasks that determines the duration of the project,
2. establish "most likely" time estimates for individual tasks by applying statistical models, and
3. calculate "boundary times" that define a time "window" for a particular task.

Time-line Chart or Gantt chart

A time-line chart or Gantt chart enables us to determine what tasks will be conducted at a given point in time. It is a popular project scheduling tool that illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Some Gantt charts also show the dependency relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line as shown here. The time-line chart help to produce *project tables*—a tabular listing of all project tasks, their planned and actual start and end dates, and a variety of related information.



2.11 Software Risk

Risk analysis and management are actions that help a software team to understand and manage uncertainty. The risks that are analyzed and managed should be derived from thorough study of the people, the product, the process, and the project. The risk mitigation, monitoring, and management (RMMM) should be revisited as the project proceeds to ensure that risks are kept up to date.

Risk concerns the future happening that always involves two characteristics: (a) *uncertainty*—the risk may or may not happen; that is, and (b) *loss*—if the risk becomes a reality, unwanted losses will occur. When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered.

- **Project risk:** threaten the project plan.

Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.

- **Technical risks:** threaten the quality and timeliness

Technical risks identify potential design, implementation, interface, verification, and maintenance problems. In addition, specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology are also risk factors. Technical risks occur because the problem is harder to solve than expected.

- **Business risks:** threaten the feasibility

It includes the top five business risks which are

  - *Market risk:* system that no one really wants
  - *Strategic risk:* product no longer fits into the overall business strategy
  - *Sales risk:* the sales force doesn't understand how to sell the product
  - *Management risk:* losing the support of senior management
  - *Budget risks:* losing budgetary or personnel commitment

- **Known risks**

Risk that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).

- **Predictable risks**

These risks are identified from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).

- **Unpredictable risks**

They can and do occur, but they are extremely difficult to identify in advance.

A. Reactive and Proactive Risks

There are two ways that software engineers can handle risk.

- *Reactive risk strategies:* Software engineer corrects a problem as it occurs or never worrying about problems until they happened. This is often called a *fire-fighting mode.*
- *Proactive risk strategies:* A considerably more intelligent strategy for risk management where a proactive software engineer starts thinking about possible risks in a project before they occur**.** A proactive strategy begins long before technical work is initiated where potential risks are identified, their probability and impact are assessed, and they are ranked by importance. Then, the software team establishes a plan for managing risk. The primary objective is to avoid risk, but all risks cannot be avoided so exercise to make them minimal. In this chapter we are solely focus on the proactive risk management strategies.

B. Risk Identification

Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.). By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary. The different categories of risk (project, technical, business, known, predictable, and unpredictable) can be further divided as:

- *Generic risks:* They are a potential threat to every software project.
- *Product-specific risks:* They can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built. To identify product-specific risks, the project plan and the software statement of scope are examined, and analyze the special characteristics of any product that may threaten the project plan.

One method for identifying risks is to create a **risk item checklist**; we need to identify the following area from where risk will appear:

- *Product size*—risks associated with the overall size of the software to be built or modified.
- *Business impact*—risks associated with constraints imposed by management or the marketplace.
- *Stakeholder characteristics*—risks associated with the sophistication of the stakeholders and the developer's ability to communicate with stakeholders in a timely manner.
- *Process definition*—risks associated with the degree to which the software process has been defined and is followed by the development organization.
- *Development environment*—risks associated with the availability and quality of the tools to be used to build the product.
- *Technology to be built*—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
- *Staff size and experience*—risks associated with the overall technical and project experience of the software engineers who will do the work.

Risk components

The risk components defined by U.S. air force are defined in the following manner:

- *Performance risk*—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.
- *Cost risk*—the degree of uncertainty that the project budget will be maintained.
- *Support risk*—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.
- *Schedule risk*—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

The impact of each risk component is divided into one of four categories—negligible, marginal, critical, or catastrophic.

C. Risk Projection and Risk Table

Risk estimation attempts to rate each risk by calculating the probability that the risk is real and the consequences of the problems associated with the risk, should it occur. During risk estimation, the first step is the prioritizing risks and hence we can allocate resources where they will have the most impact. The process of categorization of various possible risks that may appears in in project is called *risk assessment*. The analysis of nature, scope and time are main component of risk assessment.

A **risk table** provides a simple technique for risk projection. The risk table can be implemented as a spreadsheet model. This enables easy manipulation and sorting of the entries. Here, at first all risks are listed in the first column of the table. This can be accomplished with the help of the risk item checklists reference. Each risk is categorized in the second column. The probability of occurrence of each risk is entered in the next column of the table. The probability value for each risk can be estimated by team members individually.

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
Σ				
Σ				
Σ				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

Risk Category:
PS = Project size risk
BU = Business risk
TE = Technology risk
DE = Development risk
ST = Stakeholder risk
CU =Customer risk

Fig. RISK TABLE

Finally, the table is sorted by probability and by impact. High-probability, high-impact risks percolate to the top of the table, and low-probability risks drop to the bottom. This accomplishes first-order risk prioritization. After that the table is sorted and defines a cutoff line. The *cutoff line* (drawn horizontally at some point in the table) implies that only risks that lie above the line will be given further attention. Risks that fall below the line are reevaluated to accomplish second-order prioritization. The column labeled RMMM contains a pointer into a *risk mitigation, monitoring, and management plan.*

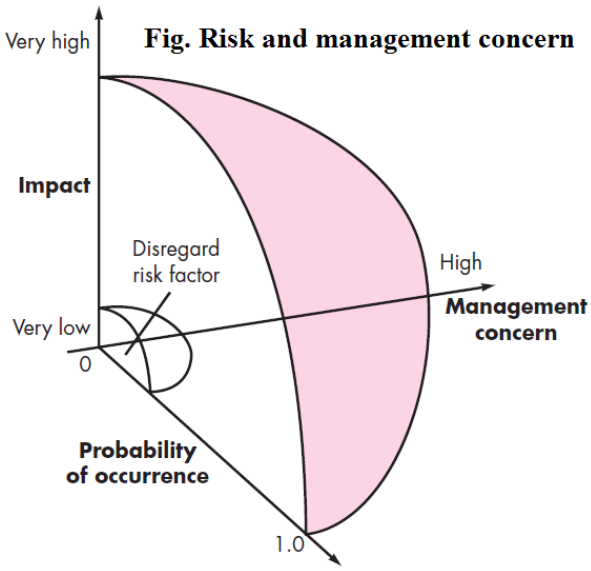


Fig. Risk and management concern

Risk impact and probability have a distinct influence on management concern. A risk factor that has a high impact but a very low probability of occurrence should not absorb a significant amount of management time. However, high-impact risks with moderate to high probability and low-impact risks with high probability should be carried forward into the risk analysis steps that follow.

Three factors affect the consequences that are likely if a risk does occur: its nature, its scope, and its timing. The overall *risk exposure* RE is determined using the following relationship: **RE = *P X C,*** Where *P* is the probability of occurrence for a risk, and *C* is the cost to the project should the risk occur.

D. Risk Refinement

During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage. One way to do this is to represent the risk in *condition-transition-consequence* (CTC) format. That is, the risk is stated in the following form:

Given that <condition> then there is concern that (possibly) <consequence>.

For example:

- **Sub-condition 1.** Certain reusable components were developed by a third party with no knowledge of internal design standards.
- **Sub-condition 2.** The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.
- **Sub-condition 3.** Certain reusable components have been implemented in a language that is not supported on the target environment.

E. Risk Mitigation, Monitoring, and Management (RMMM)

All of the risk analysis activities assist the project team in developing a strategy for dealing with risk. An effective strategy must consider three issues: risk avoidance, risk monitoring, and risk management and contingency planning.

1. Risk Mitigation (Risk Avoidance)

The risk mitigation is proactive approach to adopting always the best strategy to reduce future possibilities of risk. **For example**, assume that high staff turnover is noted as a project risk after evaluating the high risk probability and impact of risk. Now, to mitigate this risk, we would develop a strategy for reducing turnover by taking the following possible steps:

- Meet with current staff to determine causes for turnover (poor working conditions, low pay, and competitive job market).
- Mitigate those causes that are under control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organize project teams so that information about each development activity is widely dispersed.
- Define work product standards and establish mechanisms to be sure that all models and documents are developed in a timely manner.
- Conduct peer reviews of all work (so that more than one person is "up to speed").
- Assign a backup staff member for every critical technologist.

2. Risk Monitoring

As the project proceeds, *risk-monitoring* activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of **high staff turnover**, the following factors must be monitored:

- The general attitude of team members based on project pressures,
- The degree to which the team has jelled,
- Interpersonal relationships among team members,
- Potential problems with compensation and benefits, and
- The availability of jobs within the company and outside

In addition to monitoring these, other risk monitoring parameters are effectiveness of risk mitigation steps, documentation and if a newcomer were forced to join the software team somewhere in the middle of the project.

3. Risk Management

If assumes that mitigation efforts have failed and that the risk has become a reality, risk management and contingency planning is conducted. Risk is not limited to the software project itself. Risks can occur after the software has been successfully developed and delivered to the customer. These risks are typically associated with the consequences of software failure in the field.

Continuing the **example**, the project is well under way and a number of people announce that they will be leaving. If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team. In addition, to "get up to speed", newcomers enabled, and those individuals who are leaving are asked to stop all work and spend their last weeks in "knowledge transfer mode." This might include video-based knowledge capture, the development of "commentary documents or Wikis," and/or meeting with other team members who will remain on the project. The activities in risk management are:

a. Risk assignment: Figuring out what the risks are and what to focus on.
- Make the list of affecting factor for the project.
- Assign the probability of occurrences and potential loss of each items listed.
- Rank the item from most to least dangerous.

b. Risk control
- Mitigate (reduce) the highest order risk.
- Resolve the high order risk factor.
- Monitoring the effectiveness of the strategies and the challenging level of risk throughout the projects.

Software safety and hazard analysis are software quality assurance activities that focus on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.

The RMMM plan

The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan. Some software teams do not develop a formal RMMM document. Rather, each risk is documented individually using a *risk information sheet* (RIS). In most cases, the RIS is maintained using a database system so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily. The format of the RIS is illustrated in figure. Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence.

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/09	Prob: 80%	Impact: high
Description: Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.			
Refinement/context: Subcondition 1: Certain reusable components were developed by a third party with no knowledge of internal design standards. Subcondition 2: The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components. Subcondition 3: Certain reusable components have been implemented in a language that is not supported on the target environment.			
Mitigation/monitoring: 1. Contact third party to determine conformance with design standards. 2. Press for interface standards completion; consider component structure when deciding on interface protocol. 3. Check to determine number of components in subcondition 3 category; check to determine if language support can be acquired.			
Management/contingency plan/trigger: *RE* computed to be $20,200. Allocate this amount within project contingency cost. Develop revised schedule assuming that 18 additional components will have to be custom built; allocate staff accordingly. Trigger: Mitigation steps unproductive as of 7/1/09.			
Current status: 5/12/09: Mitigation steps initiated.			
Originator: D. Gagne		Assigned: B. Laster	

2.12 Software Maintenance

Maintenance corrects defects, adapts the software to meet a changing environment, and enhances functionality to meet the evolving needs of customers. The software maintenance begin almost immediately with in a day or week or month to adopt the challenges. At an organizational level, maintenance is performed by support staff that are part of the software engineering organization.

Most of software organization spend more money and time for maintaining existing programs. It is not unusual for a software organization to expend as much as 60 to 70 percent of all resources on software maintenance. Another software maintenance problem is the mobility of software people. It is likely that the software team (or person) that did the original work is no longer around. Worse, other generations of software people have modified the system and moved on. And today, there may be no one left who has any direct knowledge of the legacy system.

To reduce the software maintenance problem, we have to use the well-defined coding standards and conventions, leading to source code that is self-documenting and understandable. A variety of quality assurance techniques also uncover potential maintenance problems before the software is released.