UNIT –III

Planning a Software Project

3.1 Effort estimation

- Effort and schedule estimates are essential prerequisites for planning the project.
- Without these, even simple questions like
  - ✓ is the project late?
  - ✓ are there cost overruns? And
  - ✓ when is the project likely to complete?

  Cannot be answered.
- For software development human resources are needed. Therefore most cost estimation procedures focuses on estimating effort in terms of person-months.
- Software requirement analysis plays an important role in effort estimation.
- There are 2 popular methods for effort estimation
  1. Top down estimation approach
  2. Bottom up estimation approach

3.1.1 Top down estimation approach

- The top down estimation approach considers effort as the function of Size. The larger the project, the greater is the effort requirement.
- Let P=productivity (in KLOC/PM) then the effort estimate for the project will be SIZE/P person months (PM).
- But this approach can work only if the size and type of the project are similar.
- A more general function for determining effort from size that is commonly used is of the form

$$\textbf{Effort} = a \times \textbf{SIZE}^{b}$$

  - ✓ where a and b are constants, and
  - ✓ project size is generally in KLOC

- Values for these constants for an organization are determined through regression analysis.
- The top-down approach is normally associated with parametric models such as function points and COCOMO.
- Having calculated the overall effort required, the problem is then to allocate proportions of that effort to the various activities within that project.

Advantages

- Very useful when cost estimates are needed in the very early phases of a project.
- It can be used to estimate the effort even when you have less information available on the project.
- It allows refining your earlier estimates and adding detail as the project moves forward.

Disadvantages

- Less accurate than other estimating techniques.
- Provides less scope to get lower levels of input.

3.1.2 Bottom up estimation approach

- In this approach, the project is first divided into tasks and then estimates for the different tasks of the project are obtained.
- From the estimates of the different tasks, the overall estimate is determined.
- This type of approach is also called as **Activity-Based Estimation**.
- Once the project is partitioned into smaller tasks, it is possible to directly estimate the effort required for them, especially if tasks are relatively small.
- The procedure for estimation can be summarized as follows:
  1. Identify modules in the system and classify them as simple, medium, or complex.
  2. Determine the average coding effort for simple/medium/complex modules.

3. Get the total coding efforts using the coding effort of different types of modules and the counts for them.
4. Using the effort distribution for similar projects, estimate the effort for other tasks and the total effort.
5. Refine the estimates based on project-specific factors.

Advantages

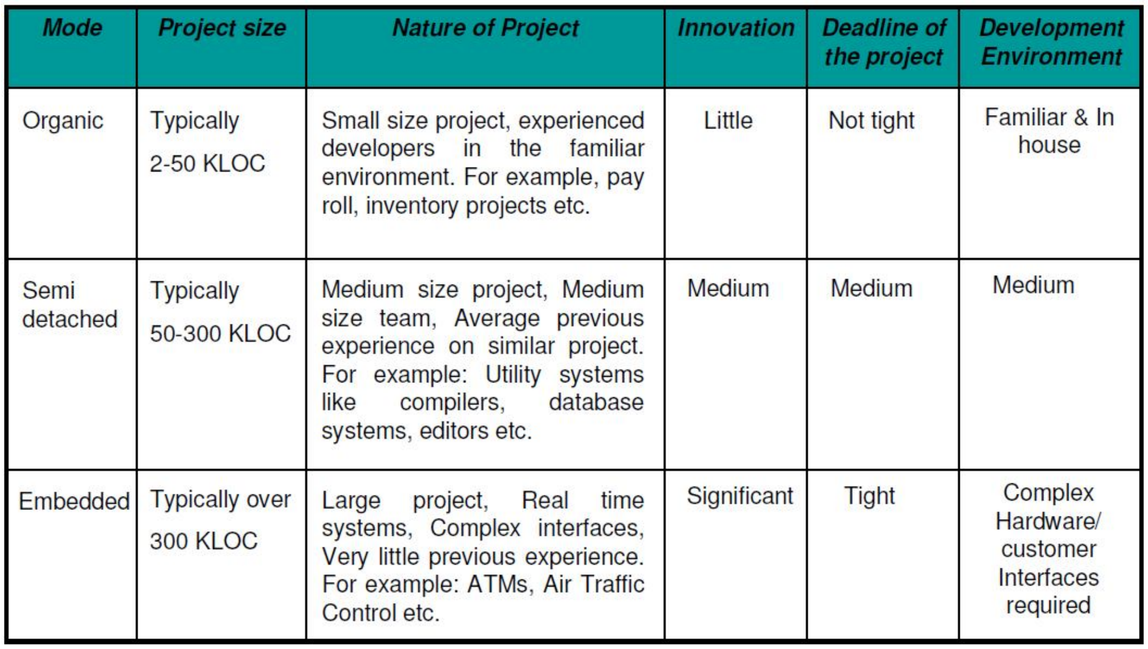
- Greater accuracy.
- It is simple and easy.

Disadvantages

- Directly estimating the overall effort is not possible, because some tasks may omit some activities.

3.1.3 COCOMO Model

- COCOMO (Constructive COst Model) is an algorithmic cost estimation technique proposed by Boehm.
- It is designed to provide some mathematical equations to estimate software projects.
- These mathematical equations are based on historical data and use project size in the form of KLOC.
- The COCOMO model is a multi-variable model and uses several variables for effort estimation.
- The value of these parameters depends on the project mode (or type) shown in the following table.



Mode	Project size	Nature of Project	Innovation	Deadline of the project	Development Environment
Organic	Typically 2-50 KLOC	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached	Typically 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/ customer Interfaces required

- There are three variations of COCOMO model.
    1. Basic COCOMO
    2. Intermediate COCOMO and
    3. Detailed COCOMO

3.1.3.1 Basic COCOMO Model

- The Basic COCOMO Model estimates effort as a function of size measured in KLOC.
- The basic COCOMO Model is very simple, quick, and applicable to small to medium organic-type projects.
- It is given as follows:

$$\textbf{Development Effort (E) = a x (KLOC)}^{b}\ \textbf{PM}$$

$$\textbf{Development Time (T) = c x (E)}^{d}\ \textbf{Months}$$

- Where a, b, c, and d are constants and these values are determined from the historical data of the past projects.
- The values of a, b, c and d for different types of projects are shown in table below.

Project Category	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Example 3.1

Assume that a system for student course registration is planned to be developed and its estimated size is approximately 10,000 lines of code. The organization is proposed to pay Rs. 25000 per month to software engineers. Compute the development effort, development time, and the total cost for product development.

Solution

The project can be considered an organic project since, its size (10 KLOC) lies between 2-50 KLOC. Thus, from the basic COCOMOmodel,

Development Effort (E) = $3.2 \times (10)^{1.05} = 35.90$ PM

Development Time (T) = $2.5 \times (35.90)^{0.38} = 9.747$ Months

Total Product Development Cost

= Development Time *Salaries of Engineers

= 9.747×25000

= Rs. 2,43,675/-

3.1.3.2 Intermediate COCOMO Model

- The basic COCOMO model determines the effort and time based on the project size.
- There are certain other factors that can affect the project size and hence the development effort and time.
- Therefore, Boehm has introduced 15 cost drivers, considering the various aspects of product development environment.
- These cost drivers are used to adjust the project complexity and are termed as effort adjustment factors (EAF).
- These cost drivers are classified as Computer Attributes, Product Attributes, Project Attributes, and Personnel Attributes.

- The intermediate COCOMO model computes software development effort as a function of the program size and a set of cost drivers. The cost drivers and their ratings are shown in Table 3.1.
- The intermediate COCOMO model estimates the initial effort using the basic COCOMO model.
- Then the EAF is calculated as the product of 15 cost drivers.
- Total effort is determined by multiplying the initial effort with the total value of EAF.
- The computation steps are summarized below.

Development effort (E):

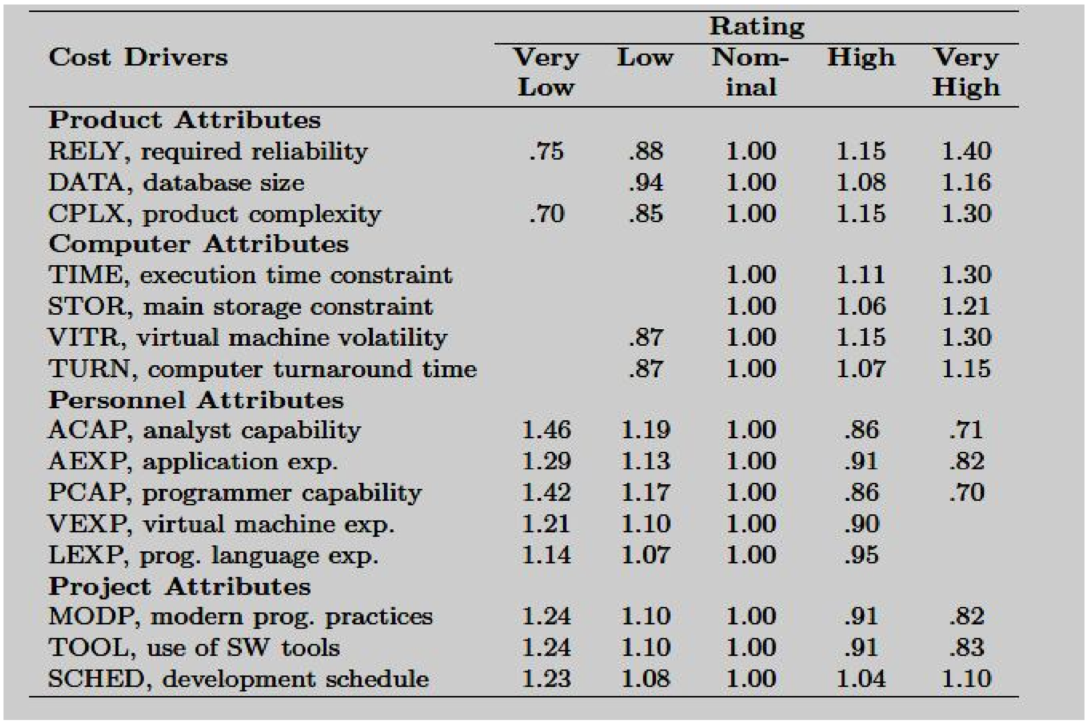
Initial effort (E_i) = $\mathbf{a \times (KLOC)^{b}}$

Effort Adjustment Factor (EAF) = $EAF_1 \times EAF_2 \times \ldots \times EAF_n$

Total development effort (E) = $E_i \times EAF$

Development time (T) = $\mathbf{c \times (E)^{d}}$

Table 3.1: Cost drivers and their values



	Rating				
Cost Drivers	**Very Low**	**Low**	**Nom-inal**	**High**	**Very High**
Product Attributes					
RELY, required reliability	.75	.88	1.00	1.15	1.40
DATA, database size		.94	1.00	1.08	1.16
CPLX, product complexity	.70	.85	1.00	1.15	1.30
Computer Attributes					
TIME, execution time constraint			1.00	1.11	1.30
STOR, main storage constraint			1.00	1.06	1.21
VITR, virtual machine volatility		.87	1.00	1.15	1.30
TURN, computer turnaround time		.87	1.00	1.07	1.15
Personnel Attributes					
ACAP, analyst capability	1.46	1.19	1.00	.86	.71
AEXP, application exp.	1.29	1.13	1.00	.91	.82
PCAP, programmer capability	1.42	1.17	1.00	.86	.70
VEXP, virtual machine exp.	1.21	1.10	1.00	.90	
LEXP, prog. language exp.	1.14	1.07	1.00	.95	
Project Attributes					
MODP, modern prog. practices	1.24	1.10	1.00	.91	.82
TOOL, use of SW tools	1.24	1.10	1.00	.91	.83
SCHED, development schedule	1.23	1.08	1.00	1.04	1.10

Example 3.2

Suppose a library management system (LMS) is to be designed for an academic institution. From the project proposal, the following five major components are identified:

Online data entry	-	1.0 KLOC
Data update	-	2.0 KLOC
File input and output	-	1.5 KLOC
Library reports	-	2.0 KLOC
Query and search	-	0.5 KLOC

The database size and application experience are very important in this project. The use of the software tool and the main storage is highly considerable. The virtual machine experience and its volatility can be kept low. All other cost drivers have nominal requirements. Use the COCOMO model to estimate the development effort and the development time.

Solution

The LMS project can be considered an organic category project. The total size of the modules is 7 KLOC. The development effort and development time can be calculated as follows:

Development effort

Initial effort (E_i) $= 3.2 \times (7)^{1.05}$

$= 24.6889$ **PM**

Effort Adjustment Factor (EAF)

$$= 1.16 \times 0.82 \times 0.91 \times 1.06 \times 1.10 \times 0.87 = 0.8780$$

Total effort (E) $= E_i * \text{EAF}$

$= 24.6889 \times 0.8780$

$= 21.6785$ **PM**

Development time (T) $= 2.5 \times (E)^{0.38}$ **Months**

$= 2.5\,(21.6785)^{0.38}$ **months**

$= 8.0469$ **months**

3.1.3.3 Detailed COCOMO Model

- The detailed COCOMO inherits all the features of the intermediate COCOMO model for the overall estimation of the project cost.
- For detailed planning and scheduling, it is necessary to assess the impact of the cost drivers in a phased manner.
- The detailed COCOMO model uses different cost drivers for each phase of the project. Phase-wise effort multipliers provide better estimates than the intermediate model.
- The detailed COCOMO model defines five life cycle phases for effort distribution:
    - Plan and Requirement,
    - System Design,
    - Detailed Design,
    - Code and Unit Test, and
    - Integration Test.
- The percentage of effort in a phase varies with the project size and the project nature. The phase-wise percentage of effort distribution for some KLOC values is shown in table 3.2.
- In the detailed COCOMO model, effort is calculated as a function of size (in terms of KLOC) and the value of a set of cost drivers according to each phase of the software life cycle.

Example

Compute the phase-wise development effort for the problem discussed in Example 3.2 above.

Solution

There are five components in the organic project discussed in Example above: Online Data Entry, Data Update, File Input and Output, Library Reports, Query and Search.

The estimated effort (E) is 21.6785 PM. The total size is 7 KLOC, which is between 2 KLOC and 32 KLOC. Thus, the actual percentage of effort can be calculated as follows:

H.KLOC DV + [(L.KLOC DV - H.KLOC DV) / (H.KLOC - L.KLOC)]*Size

Plan and Requirement (%) = 6 + (6 - 6) / (32 - 2) × 7 = 6%

Effort = 0.06 × 21.6785 PM = 1.30071 PM

System Design = 16 + (16 - 16) / (32 - 2) × 7 = 16%

Effort = 0.16 × 21.6785 PM = 3.46856 PM

Detailed Design = 24 + (26 - 24) / (32 - 2) × 7 = 25%

Effort = 0.25 × 21.6785 PM = 5.419625 PM

Code and Unit Test = 38 + (42 - 38) / (32 - 2) × 7= 39%

Effort = 0.39 × 21.6785 PM = 8.454615 PM

Integration Test = 22 + (16 - 22) / (32 - 2) × 7= 24%

Effort = 0.24 × 21.6785 PM =5.20284 PM

Table 3.2: Phase-wise Effort Distribution for Detailed COCOMO Model

Project type and size	Plan and requirement	System design	Detailed design	Code and unit test	Integration and test
Percentage-wise distribution of development effort					
Organic (2 KLOC)	6	16	26	42	16
Organic (32 KLOC)	6	16	24	38	22
Semidetached (32 KLOC)	7	17	25	33	25
Semidetached (128 KLOC)	7	17	24	31	28
Embedded (128 KLOC)	8	18	25	26	31
Embedded (320 KLOC)	8	18	24	24	34
Percentage-wise distribution of development time					
Organic (2 KLOC)	10	19	24	39	18
Organic (32 KLOC)	12	19	21	34	26

Semidetached (32 KLOC)	20	26	21	27	26
Semidetached (128 KLOC)	22	27	19	25	29
Embedded (128 KLOC)	36	36	18	18	28
Embedded (320 KLOC)	40	38	16	16	30

3.4 Project Schedule and Staffing

- Project scheduling is one of the important activities done by the project manager in order to complete the project successfully.
- Project scheduling involves planning of various activities that need to be accomplished when, how and by whom. These activities are as follows.
    - Task identification
    - Work breakdown structure
    - Task interdependency identification
    - Time allocation
    - Resource allocation
    - Monitoring, tracking and control
- For a project with some estimated effort, multiple schedules (or project duration) are possible.
- For example, if a project whose effort estimate is 56 person-months, then
    - ✓ A schedule of 8 months is possible with 7 people.
    - ✓ A schedule of 7 months with 8 people is also possible,
    - ✓ A schedule of approximately 9 months with 6 people is also possible.
    - ✓ But a schedule of 1 month with 56 people is not possible.
    - ✓ Similarly, no one would execute the project in 28 months with 2 people.
- Hence, the objective is to fix a reasonable schedule to do the project.

- One method to determine the overall schedule is to determine it as a function of effort.
- The IBM Federal Systems Division found that the total duration, M, in calendar months can be estimated by **$M = 4.1*E^{0.36}$** .
- In COCOMO, the equation for schedule for an organic type of software is **$M = 2.5*E^{0.38}$** .
- Another method for determining a schedule for medium-sized projects is the rule of thumb called the **Square Root Check**. In this, the schedule can be determined by computing the square root of the total effort in person months.
- Once the schedule is determined, then it is required to determine the number of people required in each phase of the project development.
- Usually, number of people can be utilized in a software project tends to follow the Rayleigh curve as shown below.
- It says that, in the beginning and the end, few people are needed on the project; the peak team size (PTS) is needed somewhere near the middle of the project; and again fewer people are needed after that.

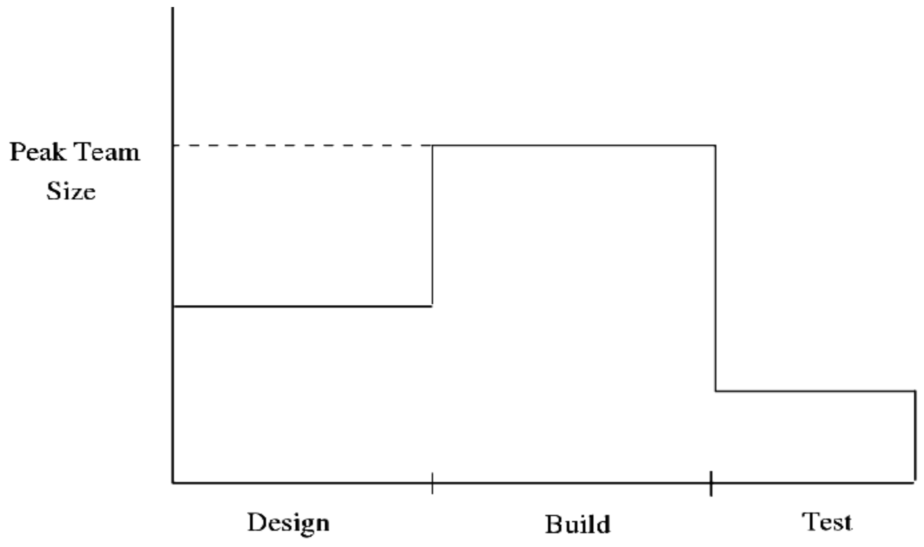


Figure 3.2: **Manpower ramp-up in a typical project**

- This occurs because only a few people can be used in the initial phases of requirements analysis and design. The human resources requirement peaks during coding and unit testing, and during system testing and integration, again fewer people are required.

- Generally speaking, design requires about a quarter of the schedule, build consumes about half, and integration and system testing consume the remaining quarter. COCOMO gives 19% for design, 62% for programming, and 18% for integration.

3.5 Quality Planning

- Software productivity and quality are the most considerable challenges in software development.
- It is easy to set goals for schedule and effort, but setting a goal for quality and then planning to meet it is very much difficult.
- Hence, often, quality goals are specified in terms of acceptance criteria.
- One acceptance criterion is the number of defects found during the acceptance testing.
    - Eg. The no.of defects found in software during acceptance testing should not exceed 100.
- Software development is a highly people-oriented activity and hence it is error-prone.
- The quality of the software product is determined based on the number of defects it has.
- Therefore, to plan for quality, let us first understand the defect injection and removal cycle because defects can be identified by injecting the known bugs into software. The defect injection and removal cycle is shown in figure 3.3.
- In a software project, we start with no defects. Then defects are injected into the software being built during different phases in the project. i.e. defects are injected in the requirements specification, the high-level design, the detailed design, and coding.

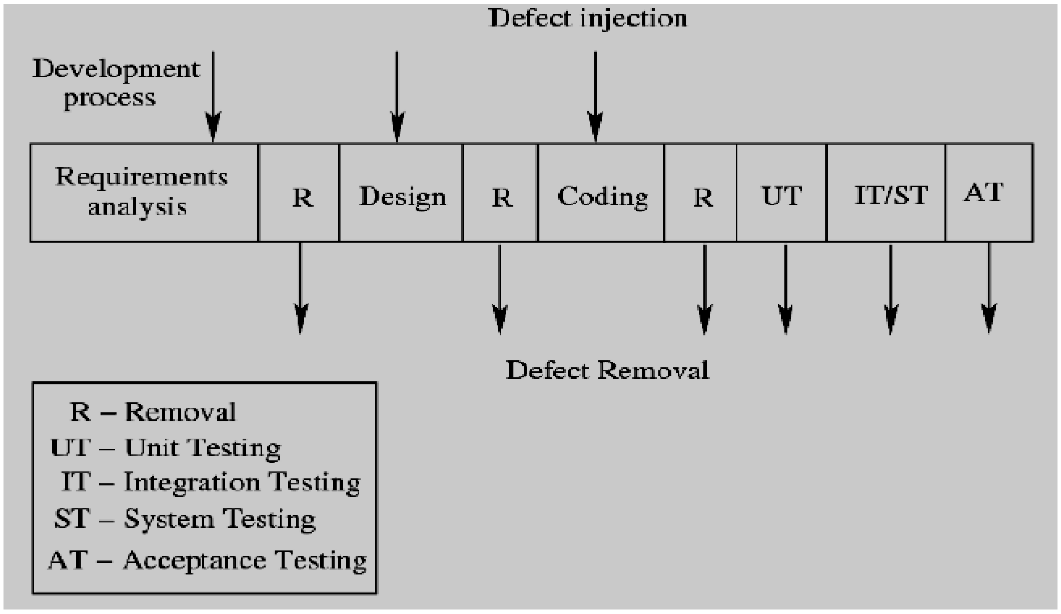


Figure 3.3: **Defect injection and removal cycle.**

- The above figure shows that defects are injected during requirements analysis, design and coding and then defects are identified and removed using QC activities.
- The QC activities for defect removal include requirements reviews, design reviews, code reviews, unit testing, integration testing, system testing, acceptance testing, etc.
- Hence, ensuring quality revolves around two main themes:
  - Reduce the defects being injected, and
  - Increase the defects being removed.
- The first is often done through standards, methodologies, following of good processes, etc., which help reduce the chances of errors by the project personnel.
- The quality plan therefore focuses mostly on planning suitable quality control tasks for removing defects.
- Reviews and testing are two most common QC activities utilized in a project to remove the defects.

- Hence, the quality plan for the project is a specification of which QC task is to be done and when, and what process and guidelines are to be used for performing the QC task.
- Typically, the QC tasks will be schedulable tasks in the detailed schedule of the project. For example, it will specify what documents will be inspected, what parts of the code will be inspected, and what levels of testing will be performed.

3.6 Risk Management

- Software projects and businesses are full of uncertainties due to changing requirements, varying range of applications, informal processes and technological advancements.
- Risk is an unfavourable situation that may lead to undesirable outcome.
- Risk management is a proactive approach for minimizing the uncertainty and potential problems associated with a project.
- Software risk management is necessary
  - To reduce the rework caused by missing, erroneous, or ambiguous requirements, design or code.
  - To avoid software project disasters including overrun of budgets and schedules, defect-driven software and operational failures.
  - To stimulate a win-win software solution where customers receive the product they need and the vendors make the profits they except.
  - To keep provision for the detection and prevention of risks.
- There are typically 3 types of risk occurrences.
  1. **Project risks** – are concerned with project planning and scheduling. Most of the projects run behind schedule, become over budget and have lack of skilled engineers.
  2. **Product risks** – are concerned with design and development of software products. These risks affect product quality and its

performance. These risks include technology obsolescence, changing requirements, technical uncertainty, excessive constraints, lack of technical knowledge etc.

3. **Business risks** – are the negative impacts on the operation or profitability of an organization. Eg. An organization is unable to produce products which are demanded in the market; the competitor has launched an alternative product etc. Business risks are mainly due to wrong decisions.

3.6.1 Risk Management Activities

- In order to handle all the above risks, some systematic process of risk management is required. The risk management process includes the following risk management activities.

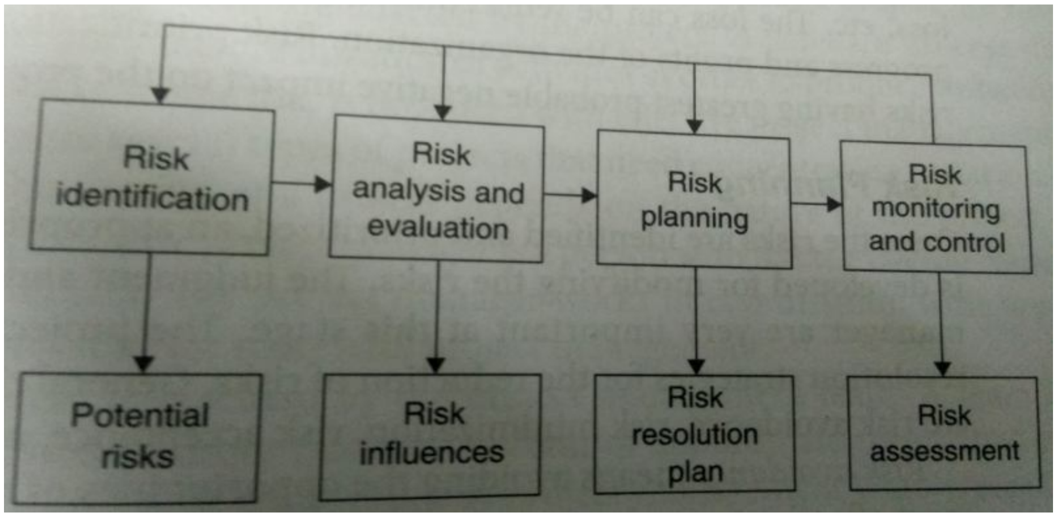


Figure 3.4: **Risk Management Activities**

Risk Identification

- In this potential risks that affect the software product are identified along with their sources, root causes, consequences and types.
- Different types of risks may arise. Eg. Requirements risks, technology risks, organizational risks, tool risks, human resources risks, estimation risks etc.

- Different techniques are used to identify the risks, such as interviewing, brainstorming, assumption analysis, critical path analysis, utilization of risk taxonomies etc.
- Risk identification is done based on the historical data.

Risk Analysis

- In this each risk is analyzed independently by examining and assessing its impact, probability, and seriousness.
- It can be done using different techniques like metrics, decision trees and scenario analysis.
- The list of risks is then grouped and prioritized/ranked based on the risk analysis. It helps in resource allocation and management.
- Boehm defined the risk exposure (RE) to establish risk priorities. It measures the impact of a risk in terms of loss.
- RE is defined as the probability of an undesired outcome times the expected loss if that outcome occurs.

  RE = Prob(UO) * Loss(UO)

  Where Prob(UO) is the probability of occurrence of the undesirable outcome and

  Loss(UO) is the total loss incurred due to that undesirable outcome.

Risk Planning

- Once the risks are identified and prioritized, an appropriate management policy is developed to handle the risks.
- The project manager uses risk resolution strategies for reducing the risks. These are
- **Risk avoidance** – avoiding the risks. It is the most effective way. Project managers should not take decisions of working in a new way that may increase the risk.
  - Eg. Not paying money by debit card.
- **Risk minimization** – focus on identifying the options for reducing the likelihood or consequences of the risk.

  - Eg. Backup programmers must be recruited to continue the project in case of people working on the project leave the organization.

- **Risk acceptance** – comes when you have no option other than accepting the risk. In this case, project manager takes executive support to mitigate the risk.
- **Risk transfer** – transfer the risk to somebody else. This can be done by choosing outsourcing, subcontracting, buying a resource or tool.

Risk Monitoring and Control

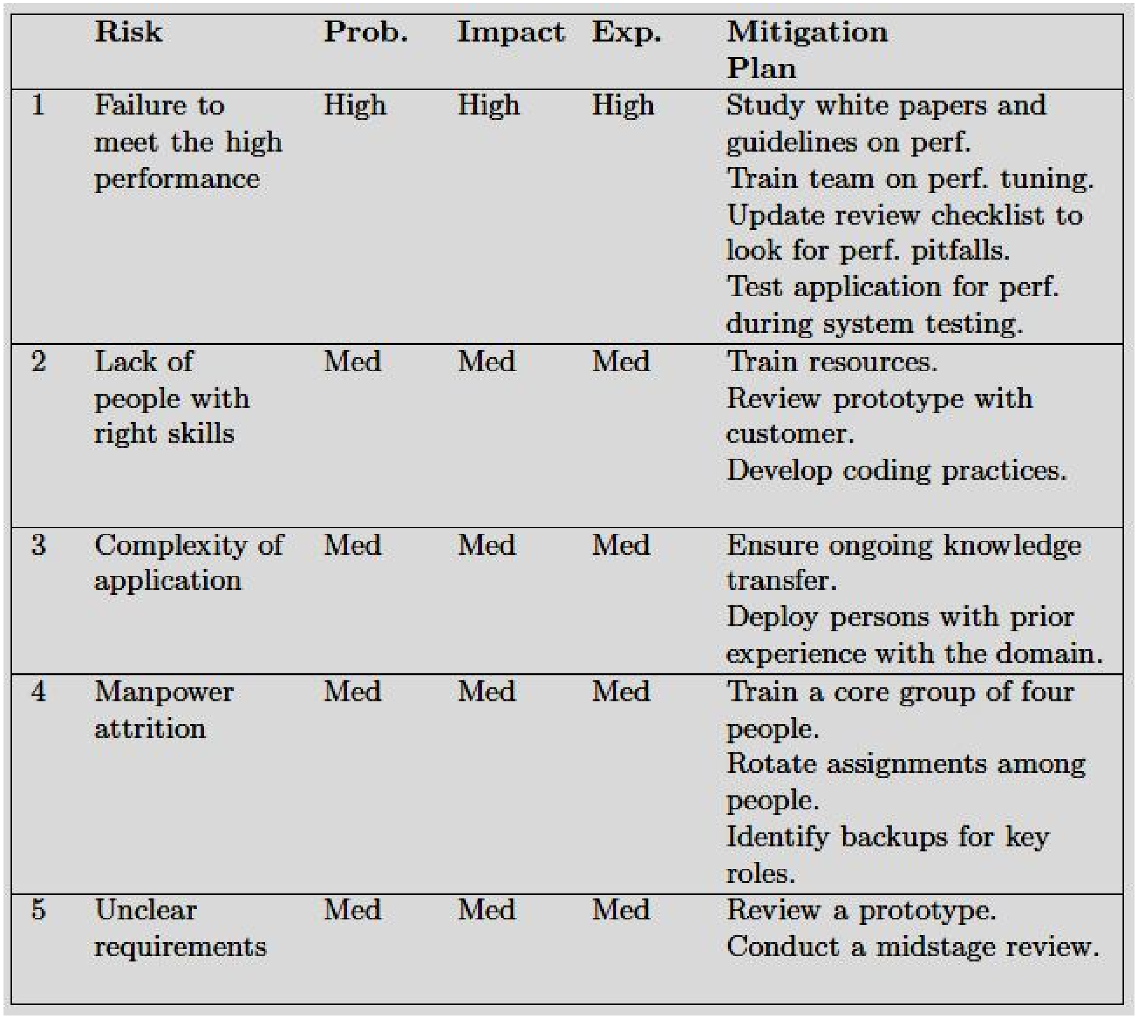
- Risk monitoring and control ensures new risks are detected and managed.
- Risk action plans are implemented to reduce the impact of risks.
- Policies and standards are regularly carried out to identify the opportunities for improvement.
- If needed, changes are made in the organizational environment to cope with risks.

3.6.2 A Practical Risk Management Planning Approach

- In this approach, the probability of a risk occurring is categorized as low, medium, or high. The risk impact can also be classified as low, medium, and high. With these ratings, the following simple method for risk prioritization can be specified:

1. For each risk, rate the probability of its happening as low, medium, or high.
2. For each risk, assess its impact on the project as low, medium, or high.
3. Rank the risks based on the probability and effects on the project; for example, a high-probability, high-impact item will have higher rank than a risk item with a medium probability and high impact. In case of conflict, use judgment.
4. Select the top few risk items for mitigation and tracking.

- The following figure shows a practical risk management plan for a typical project.



	Risk	Prob.	Impact	Exp.	Mitigation Plan
1	Failure to meet the high performance	High	High	High	Study white papers and guidelines on perf. Train team on perf. tuning. Update review checklist to look for perf. pitfalls. Test application for perf. during system testing.
2	Lack of people with right skills	Med	Med	Med	Train resources. Review prototype with customer. Develop coding practices.
3	Complexity of application	Med	Med	Med	Ensure ongoing knowledge transfer. Deploy persons with prior experience with the domain.
4	Manpower attrition	Med	Med	Med	Train a core group of four people. Rotate assignments among people. Identify backups for key roles.
5	Unclear requirements	Med	Med	Med	Review a prototype. Conduct a midstage review.

Figure 3.5: Practical risk management plan for a project