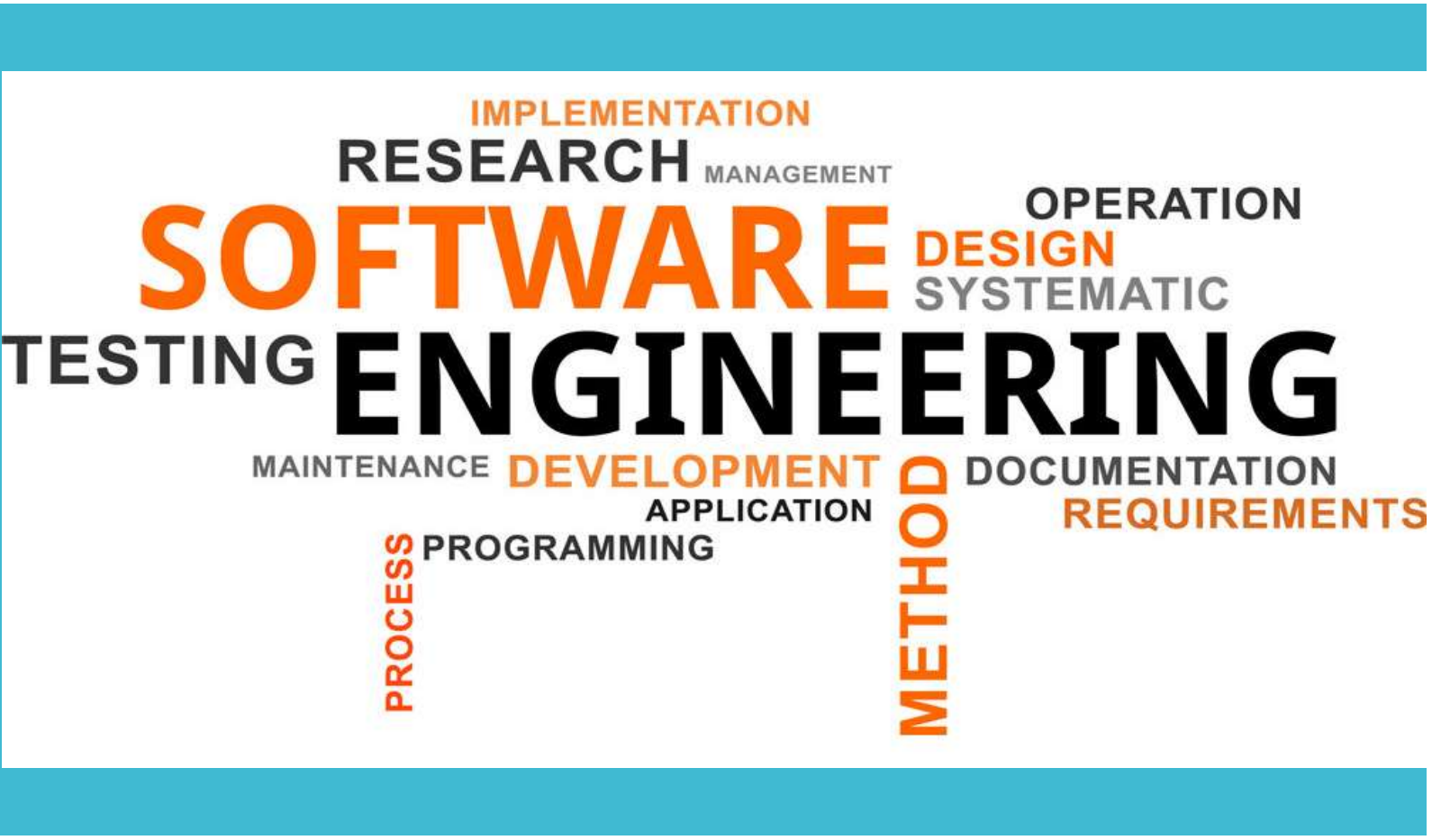


Marwadi University
Marwadi Chandarana Group
NAAC A+

Department of
Computer Engineering

Chapter 2
Project Management Concepts

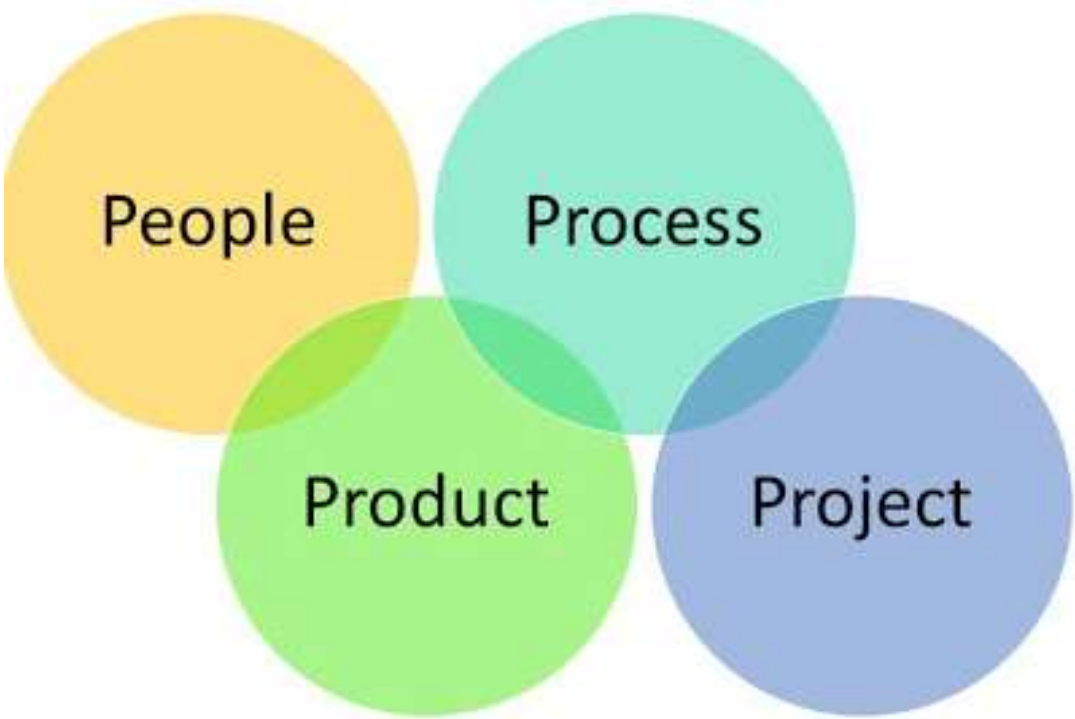
Subject Code: 01CE0607

Prof. Urvi Dhamecha

Contents

- The Management Spectrum
- 4P's(The People, The Project, The Product, The Process)
- The W5HH Principle
- Software Metrics
  - Process and Project Metrics
  - Product Metrics
  - Measures, Metrics and Indicators
- Software Measurement
- Metrics for Software Quality
- Empirical Estimation Models
- Scheduling
- Risk Strategies: Reactive versus proactive
- Risk Management Process: Risk Identification, Risk Projection, Risk Refinement
- RMMM Plans
- Safety Risks and Hazards

Management Spectrum



Management Spectrum

1. People
   - People in Software engineering process is an important part.
   - Organizations that achieve high levels of maturity in the people management area have a higher likelihood of implementing effective software engineering practices.
2. Product
   - The outcome of a software project.
   - All the outputs that are produced while the activities are being executed.

Management Spectrum

3. Process

- Specifies activities related to production of software.
- Specifies the abstract set of activities that should be performed to go from user needs to final product.

4. Project

- Software development work in which a software process is used.
- The actual act of executing the activities for some specific user needs.

People

Stakeholders :-

1. Project managers: Plan, Motivate, Organize and control the project work.
2. Senior Managers: Define Business issues and have significant influence on project.
3. Practitioners: Deliver all technical skill
4. Customers: who satisfy the requirement.
5. End Users: who interact with software produc

People

Team Leaders:

Key traits of an effective project manager are :

1. Problem Solving
2. Managerial identity
3. Achievement
4. Influence and team building

People

The Software team:

Factors which should be considered when planning the structure of team are:

1. Difficulty of problem to be solved
2. "Size" of the resultant programs in lines of code or function points.
3. Time that team will stay together
4. Degree to which problem can be modularized
5. Required quality and reliability of the system to be built
6. Rigidity of the delivery date
7. Degree of sociability required for the project

People

Constantine suggests four organizational paradigms for software engineering teams:

1. A closed paradigm
2. A random paradigm
3. An open paradigm
4. A synchronous paradigm

People

- To achieve a high performance team:
- Team members must have trust in one another
- The distribution of skills must be appropriate to the problem.
- Mavericks may have to be excluded from the team, if team cohesiveness has to be maintained.
- A jelled team is a group of people so strongly knit that the whole is greater than the sum of the parts.

People

- Factors that foster a potentially toxic team environment:
- A frenzied work atmosphere
- High frustration that causes friction among team members
- A fragmented or poorly coordinated software process
- An unclear definition of roles on the software team
- Continuous and repeated exposure to failure.

Product

Scope of the product must be established and bounded. It must be unambiguous and understandable at the management and technical levels.

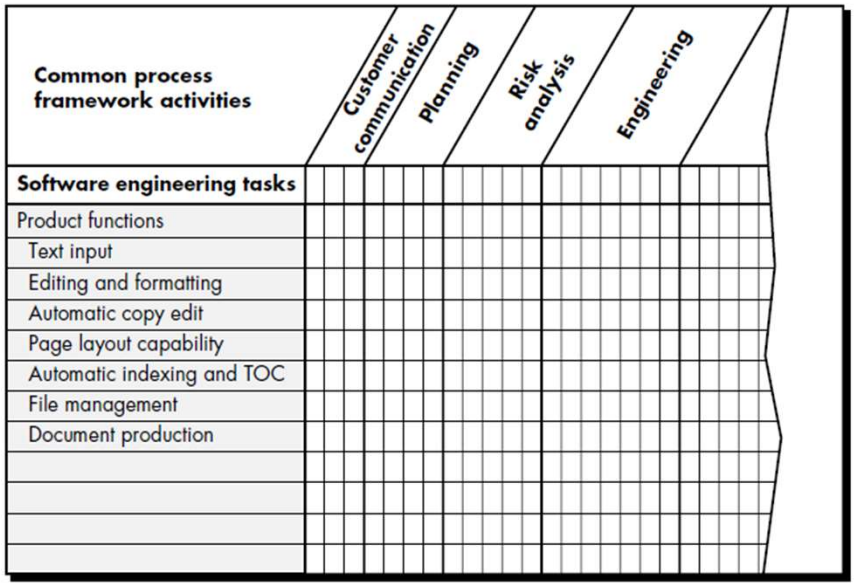
Software scope:

- Context
- Information objectives
- Function and Performance

Problem decomposition

Process

- Melding the Product and the Process



- Process Decomposition

Project

- Start on the right foot
- Maintain momentum
- Track progress
- Make smart decisions
- Conduct a post-mortem analysis

W5HH of Project Management

- Barry Boehm suggests an approach that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources.
- W5HH principle, after a series of questions that lead to a definition of key project characteristics and the resultant project plan.

1. Why is the system being developed?
   - Focus on business reason & problem statements.
   - Enables all parties to assess the validity of business reasons for the software work.

W5HH of Project Management

2. What will be done?
   - Identify key project tasks and the milestones that are required by the customer.
3. When will it be completed?
   - To help the team to establish a project schedule
4. Who is responsible for each activity?
   - The role and responsibility of each member of the software team must be defined.

W5HH of Project Management

5. Where are they organizationally located?
   - Not all roles and responsibilities reside within the software team itself. The customer, users, and other stakeholders also have responsibilities.
6. 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.
7. How much of each resource is needed?
   - Develop estimation

- It is applicable regardless of size or complexity of software project

Terminologies

- Measure
  - It provides a quantitative indication of the range, amount, dimension, capacity or size of some attributes of a product or process.
    - Ex., the number of uncovered errors.
- Metrics
  - It is a quantitative measure of the degree that any attribute belongs to system, component or process.
  - It relates individual measures in some way.
    - Ex., number of errors found per review.

Terminologies

- Indicators
  - A metric or combination of metrics that provides insight into the software process, project or the product itself.
  - It enables the project manager or software engineers to adjust the process, the project or the product to make things better.
    - Ex., Product Size is an indicator of increased coding, integration and testing effort

Terminologies

- Direct Metrics
  - Immediately measurable attributes.
    - Ex., Line of Code (LOC), Execution Speed, Defects Reported
- Indirect Metrics
  - Aspects that are not immediately quantifiable.
    - Ex., Functionality, Quantity, Reliability
- Faults
  - Errors: Faults found by the practitioners during software development.
  - Defects: Faults found by the customers after release

Why Measure Software?

- To determine quality of product and process.
- To predict quality of product and process.
- To improve quality of product and process.

Matric Classification

- Process: Define activities related to the production of software.
- Project: Software development work in which a software process is used.
- Product: The outcome of software project.

Process Metrics

- Process indicators enable a software engineering organization to gain insight into the effectiveness of an existing process (i.e., the paradigm, software engineering tasks, work products, and milestones).
- They enable managers and practitioners to assess what works and what doesn't.
- Process metrics are collected across all projects and over long periods of time. Their intent is to provide indicators that lead to long-term software process improvement.

Process Metrics

- Process Metrics are an invaluable tool for companies to monitor, evaluate and improve their operational performance across the enterprise
- They are used for making strategic decisions

Process Metrics

- Their intent is to provide a set of process indicators that lead to long-term software process improvement

Ex., Defect Removal Efficiency (DRE) metric

Relationship between errors (E) and defects (D)

The ideal is a DRE of 1

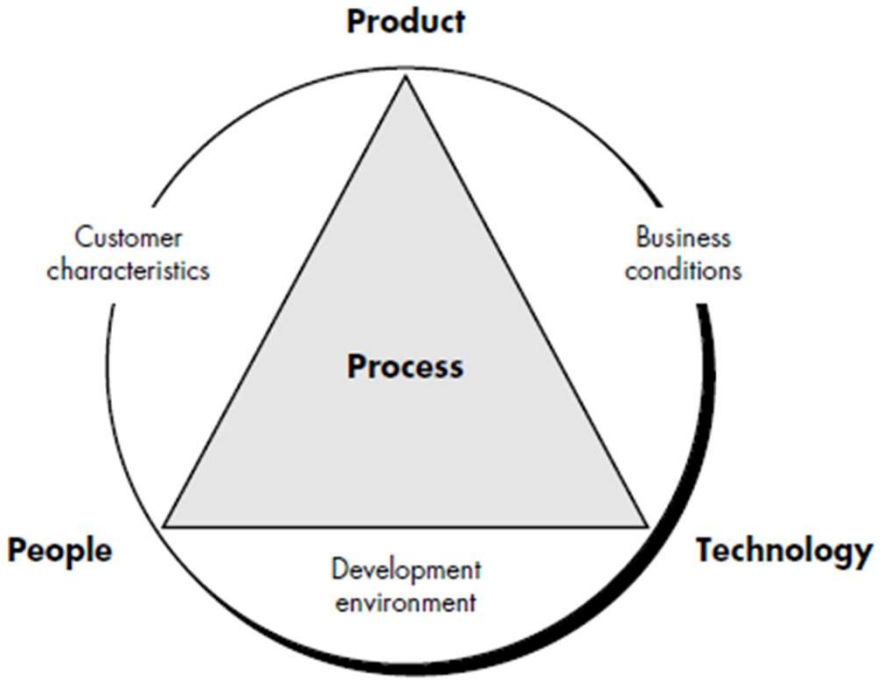
DRE = E/ (E+D)

Process Metrics

We measure the effectiveness of a process by deriving a set of metrics based on outcomes of the process such as,

1. Errors uncovered before release of the software
2. Defects delivered to and reported by the end users
3. Work products delivered
4. Human effort expended
5. Calendar time expended
6. Conformance to the schedule
7. Time and effort to complete each generic activity

Process Metrics



Determinants for software quality and organizational

Project Metrics

- Project metrics enable a software project manager to,
  1. Assess the status of an ongoing project
  2. Track potential risks
  3. Uncover problem areas before their status becomes critical
  4. Adjust work flow or tasks
  5. Evaluate the project team's ability to control quality of software work products

Project Metrics

- Metrics collected from past projects are used as a basis from which effort and time estimates are made for current software work.
- As a project proceeds, measures of effort and calendar time expended are compared to original estimates
- Project metrics are used to
  - Minimize the development schedule by making the adjustments necessary to avoid delays and mitigate (to reduce) potential (probable) problems and risks
  - Assess (evaluates) product quality on an ongoing basis and guides to modify the technical approach to improve quality

Product Metrics

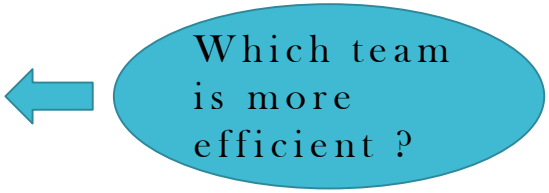
- Product metrics help software engineers to gain insight into the design and construction of the software they build.
- By focusing on specific, measurable attributes of software engineering work products.
- Product metrics provide a basis from which analysis, design, coding and testing can be conducted more objectively and assessed more quantitatively.

Types of Software Measurement

- Direct measure of the software process include cost and effort applied.
- Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.
- The cost and effort required to build software, the number of lines of code produced, and other direct measures are relatively easy to collect and measure.

Types of Software Measurement

- Indirect measures of the product that include functionality, complexity, efficiency, reliability, maintainability etc.
- The quality and functionality of software or its efficiency or maintainability are more difficult
  - Team A found : 342 errors
  - Team B found : 184 errors



- It is depends on size or complexity (i.e. functionality) of the projects.

Software Measurement

- Metrics for Software Cost and Effort estimations

1. Size-Oriented Metrics
2. Function-Oriented Metrics
3. Object-Oriented Metrics
4. Use-Case–Oriented Metrics

Size-Oriented Metrics

- Derived by standardizing quality and/or productivity measures by considering the size of the software produced.
- Thousand lines of code (KLOC) are often chosen as the normalization value.
- A set of simple size-oriented metrics can be developed for each project
  - Errors per KLOC (thousand lines of code)
  - Defects per KLOC
  - $ per KLOC
  - Pages of documentation per KLOC

Size-Oriented Metrics

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

Size-Oriented Metrics

- In addition, other interesting metrics can be computed, like
  - Errors per person-month
  - KLOC per person-month
  - $ per page of documentation
- Size-oriented metrics are not universally accepted as the best way to measure the software process.
- Opponents argue that KLOC measurements
  - Are dependent on the programming language that it penalize well-designed but short programs
  - Cannot easily accommodate nonprocedural languages

Function Oriented Metrics

- Function-oriented metrics use a measure of the functionality delivered by the application as a normalization value
- Most widely used metric of this type is the Function Point
- FP metric can be used to
  - Estimate the cost or effort required to design, code, and test the software
  - Predict the number of errors(testing)
  - Forecast the number of components and/or the number of projected source lines in the implemented system.

Computation of function points

		Weighting factor			
Measurement parameter	**Count**	**Simple**	**Average**	**Complex**	
Number of user inputs		× 3	4	6	=
Number of user outputs		× 4	5	7	=
Number of user inquiries		× 3	4	6	=
Number of files		× 7	10	15	=
Number of external interfaces		× 5	7	10	=
Count total					

Computation of function points

$$FP = \text{count total} * [0.65 + 0.01 * \Sigma(Fi)]$$

- Count Total is the sum of all FP entries obtained from previous table
- Weighting factor is determined for each organization via empirical (based on experience) data.
- Fi (i=1 to 14) are complexity value adjustment factors (VAF).

Function Point Parameters

1. Number of user inputs- Each user input that provides distinct application oriented data to the software is counted.
   - Inputs should be distinguished from inquiries, which are counted separately.
2. Number of user outputs- Each user output that provides application oriented information to the user is counted.
   - In this context output refers to reports, screens, error messages, etc.

Function Point Parameters

3. Number of user inquiries- An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted. (i.e. search index)
4. Number of files- Each logical master file (i.e. large database or separate file) is counted.
5. Number of external interfaces- All machine readable interfaces (e.g., data files on storage media) that are used to transmit information to another system are counted.

Complexity Adjustment Values Factors

- Complexity Adjustment Values (Fi) are rated on a scale of 0 (not important) to 5 (very important):
  - Does the system need reliable backup and recovery?
  - Are data communications required?
  - Are there distribute processing functions?
  - Is the performance of the system critical?
  - Can the system be able to run in an existing, heavily, and largely utilized operational environment?
  - Does the system require on-line data entry?
  - Does the input transaction is required by the on-line data entry to be built over multiple screens or operations?
  - Are the master files updated on-line?
  - Are the inputs, outputs, files, or inquiries complex?
  - Is the internal processing complex?
  - Is the code which is designed to be reusable?
  - Are conversion and installation included in the design?
  - Is the system designed for multiple installations in various organizations whenever required?
  - Is the application designed to facilitate or make the change and provide effective ease of use by the user?

Example-1

- User Input: 3
- User Output: 2
- User Inquiry: 2
- Internal Logical File: 1
- External Interface File: 4

Information Domain Value	Count		Weighting factor Simple	Average	Complex		
External Inputs (EIs)	3	×	(3)	4	6	=	9
External Outputs (EOs)	2	×	(4)	5	7	=	8
External Inquiries (EQs)	2	×	(3)	4	6	=	6
Internal Logical Files (ILFs)	1	×	(7)	10	15	=	7
External Interface Files (EIFs)	4	×	(5)	7	10	=	20
Count total	→						50

Example-1

- Used Adjustment Factors and assumed values are,
  - Distributed processing = 5
  - High performance= 4
  - Complex internal processing = 3
  - Code to be reusable= 2
  - Multiple sites= 3
- **Value Adjustment Factor (VAF)**

= 5+4+3+2+3 = 17

Adjusted FP

= Count Total * [0.65 + 0.01 * ∑(Fi)]

= Unadjusted FP * [0.65 + (0.01 * Adjustment Factor)]

= 50 * [0.65 + (0.01 x 17)]

= 50 x [0.82]

= 41

Function Oriented Metrics

- Advantages
  - FP is programming language independent
  - FP is based on data that are more likely to be known in the early stages of a project, making it more attractive as an estimation approach
- Disadvantages
  - FP requires some "sleight of hand" because the computation is based on subjective data
  - Counts of the information domain can be difficult to collect
  - FP has no direct physical meaning, it's just a number

Object-Oriented Metrics

- Conventional software project metrics (LOC or FP) can be used to estimate object-oriented software projects
- However, these metrics do not provide enough granularity (detailing) for the schedule and effort adjustments that are required as you iterate through an evolutionary or incremental process
- Lorenz and Kidd suggest the following set of metrics for OO projects
  - Number of scenario scripts
  - Number of key classes (the highly independent components)
  - Number of support classes
  - Average no. of support classes per key class
  - Number of subsystems

Use Case Oriented Metrics

- Like FP, the use case is defined early in the software process, allowing it to be used for estimation before significant (valuable) modeling and construction activities are initiated.
- Use cases describe (indirectly, at least) user-visible functions and features that are basic requirements for a system.
- The use case is independent of programming language, because use cases can be created at vastly different levels of abstraction, there is no standard "size" for a use case.
- Without a standard measure of what a use case is, its application as a normalization measure is suspect (doubtful).

Metrics for software quality

- The overriding goal of software engineering is to produce a high-quality system, application or product.
- The quality of a system is only as good as the requirements that describe the problem, the design that models the solution, the code that leads to an executable program, and the tests that exercise the software to uncover errors.
- Metrics such as work product (e.g., requirements or design) errors per function point, errors uncovered per review hour, and errors uncovered per testing hour provide insight into the efficiency of each of the activities implied by the metric.
- Error data can also be used to compute the defect removal efficiency (DRE) for each process framework activity.

Metrics for software quality

- There are many measures of software quality: correctness, maintainability, integrity, and usability provide useful indicators for the project team.

1. Code Quality
2. Reliability
3. Performance
4. Usability
5. Correctness
6. Maintainability
7. Integrity
8. Security

Metrics for software quality

1. Code Quality –

- Code quality metrics measure the quality of code used for the software project development.
- Maintaining the software code quality by writing Bug-free and semantically correct code is very important for a good software project development.
- In code quality both Quantitative metrics like number of lines, complexity, functions, rate of bugs generation etc and Qualitative metrics like readability, code clarity, efficiency, maintainability etc are measured.

Metrics for software quality

2. Reliability –

- Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR)

3. Performance –

- Performance metrics measures the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.

4. Usability –

- Usability metrics checks whether the program is user friendly or not. Each software is used by the end user. So it is important to measure that the end user is happy or not by using this software.

Metrics for software quality

5. Correctness –

- A program must operate correctly or it provides little value to its users. Correctness is the degree to which the software performs its required function

6. Maintainability –

- Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change in requirements.
- A simple time-oriented metric(indirect measure) is mean-time-to-change (MTTC), the time it takes to analyze the change request, design an appropriate modification, implement the change, test it, and distribute the change to all users.

Metrics for software quality

7. Integrity –

- This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security.

8. Security –

- Security metrics measures how much secure the software is ?
- Security assures that there no unauthorized changes, no fear of cyber attacks etc when the software product is in use by the end user.

Empirical Estimation Models

- **Constructive Cost Model (COCOMO)**

COCOMO Model

- COCOMO (Constructive Cost Estimation Model) was proposed by Boehm
- Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e number of Lines of Code.
- It is a procedural cost estimate model for software projects.
- Use process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time and quality.
- Boehm suggested that estimation of project parameters should be done through three stages:

1. Basic COCOMO
2. Intermediate COCOMO
3. Complete COCOMO.

COCOMO Model

- **Software Development Project Classification**

Based on the development complexity

1. Organic
2. Semidetached
3. Embedded

COCOMO- Organic type of project

- A development project is said to be of organic type, if
  - The project deals with developing a **well understood application**
  - The development team is **small**
  - The team members have **prior experience** in working with similar types of projects
  - Application programs(e.g. data processing programs)

COCOMO-Semidetached type of project

- A development project can be categorized as semidetached type, if
  - The team consists of some experienced as well as inexperienced staff
  - Team members may have some experience on the type of system to be developed
  - Utility programs (e.g Compilers, linkers)

COCOMO-Embedded type of project

- Embedded type of development project are those, which
  - Aims to develop a software strongly related to **machine hardware**
  - Team size is usually **large**
  - System programs (e.g Operating systems, real-time systems)

Comparison of COCOMO Models

Aspects	Organic	Semidetached	Embedded
Project Size	2 to 50 KLOC	50-300 KLOC	300 and above KLOC
Complexity	Low	Medium	High
Team Experience	Highly experienced	Some experienced as well as inexperienced staff	Mixed experience, includes experts
Environment	Flexible, fewer constraints	Somewhat flexible, moderate constraints	Highly rigorous, strict requirements
Effort Equation	$E = 2.4(400)^{1.05}$	$E = 3.0(400)^{1.12}$	$E = 3.6(400)^{1.20}$
Example	Simple payroll system	New system interfacing with existing systems	Flight control software

Basic COCOMO Model

- The basic COCOMO model helps to obtain a rough estimate of the project parameters.

$$E = a * (KLOC)^b \text{ Person Months}$$

$$D = 2.5 * (Effort)^c \text{ Months}$$

- **KLOC** = Kilo Lines of Code,
- **a, b** are constants for each category of software products
- **D** = Development time in months
- **E=** Effort expressed in person months (PMs).

Basic COCOMO Model

Software project	*a*	*b*	*c*
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

Basic COCOMO Model Example

- Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

$$E=a*(KLOC)^{b}\ PM$$

$$D=2.5*(efforts)^{c}\ Months$$

Estimated Size of project= 400 KLOC

(i)Organic Mode

$$E = 2.4 * (400)^{1.05} = 1295.31\ PM$$

$$D = 2.5 * (1295.31)^{0.38} = 38.07\ M$$

Basic COCOMO Model Example

(ii)Semidetached Mode

$$E = 3.0 * (400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5 * (2462.79)^{0.35} = 38.45 \text{ M}$$

(iii) Embedded Mode

$$E = 3.6 * (400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5 * (4772.8)^{0.32} = 38 \text{ M}$$

Basic COCOMO Model Example

- **For a given project was estimated with a size of 200 KLOC. Calculate the Effort, Scheduled time for development. Also, calculate the Average staff size and Productivity of the software for semidetached mode.**

- **Formula:**
- **Average Staff Size=E/D Persons**
- **Productivity=KLOC/E (KLOC/PM)**

Basic COCOMO Model Example

- **Assume that the size of an organic software product has been estimated to be 32,000 lines of source code. Assume that the average salary of software be Rs. 15,000/- month. Determine the effort required to develop the software product, the nominal development time, cost and staff size.**

Intermediate COCOMO model

- The basic COCOMO model considers that effort and development time depends only on the size of the software.
- There are many other project parameters that influence the development process.
- The intermediate COCOMO take those other factors into consideration by **defining a set of 15 cost drivers (multipliers)**
- Each of the 15 such attributes can be rated on a six-point scale ranging from "very low" to "extra high" in their relative order of importance.

Cost drivers for Intermediate COCOMO

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Intermediate COCOMO model

Effort = a x $(KLOC)^b$ * **EAF** person Month

EAF= Effort Adjustment Factor

Time of Development = 2.5 x $(Effort)^c$ Months

- The product of all cost drivers results in an effort adjustment factor (EAF).
- They are used because they are statistically significant to the cost of the project and they are not correlated to the project size (KLOC).

Intermediate COCOMO model

Intermediate COCOMO	a	b	c
Organic	3.2	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	2.8	1.20	0.32

Intermediate COCOMO model Example

- **A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers (1) with very high application experience and very little experience in the programming language being used (2) developers of very low application experience but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool.**

Complete COCOMO Model

A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity

- Most large systems are made up several smaller sub-systems
- These sub-systems may have widely different characteristics
- E.g., some sub-systems may be considered as organic type, some semidetached, and some embedded
- Also for some subsystems the reliability requirements may be high, for some the development team might have no previous experience of similar development etc.

Complete COCOMO Model

The complete COCOMO Model considers these difference in characteristics of the subsystem and estimates the effort and development time as the sum of the estimates for the individual subsystems.

- The cost of each subsystem is estimated separately.
- This approach reduces the margin of error in the final estimate.

Advantages of COCOMO Model

- Easy to estimate the total cost of the project.
- Easy to use and documented properly.
- Easy to implement with various factors.
- Data gathered from previous projects are helpful to determine the value of constants of the model.

Disadvantage of COCOMO Model

- COCOMO uses KLOC, which is not a proper measure of a program's size.
- COCOMO was proposed in 1981 keeping the waterfall model of project life cycle in mind. It fails to address other popular approaches like prototype, incremental, spiral, agile models.
- It ignores hardware and customer related issues,
- It ignores software safety and security issues.
- It is silent about the involvement and responsiveness of customer.
- It does not give proper importance to software requirements and specification phase.

Scheduling

- Project - task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when.
- Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.

Scheduling Principles

- **Compartmentalization**. The project must be compartmentalized into a number of manageable activities and tasks.
  - To accomplish compartmentalization, both the product and the process are decomposed
- **Interdependency**. The interdependency of each compartmentalized activity or task must be determined.
  - Some tasks must occur in sequence while others can occur in parallel.
  - Some activities cannot commence until the work product produced by another is available. Other activities can occur independently.

Scheduling Principles

- **Time Allocation:** each task must be assigned a start date and a completion date that are a function of the interdependencies
- **Effort validation:** Project manager must ensure that on any given day there are enough staff members assigned to complete the tasks within the time estimated in the project plan
- **Defined responsibilities:** Every task that is scheduled should be assigned to a specific team member.
- **Defined outcomes**: Every task that is scheduled should have a defined outcome.

Scheduling Principles

- **Defined milestones:** Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

- Each of these principles is applied as the project schedule evolves.

Scheduling

- To schedule the project activities, a project manager do the following:

1. Identify all the tasks needed to complete the project.
2. Break down large tasks into small activities.
   (Work Breakdown Structure)
3. Determine the dependency among different activities.
   (Activity Network)
4. Establish the most likely estimates for the time durations necessary to complete the activities.
5. Allocate resources to activities.
6. Plan the starting and ending dates for various activities.
7. Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Work Breakdown Structure

- WBS provides a notation for representing the major tasks need to be carried out in order to solve a problem.
- WBS is used to decompose a given task set recursively into small activities.
- Each node of the tree is recursively decomposed into smaller sub-activities until at the leaf level, the activities requires approximately two weeks to develop.

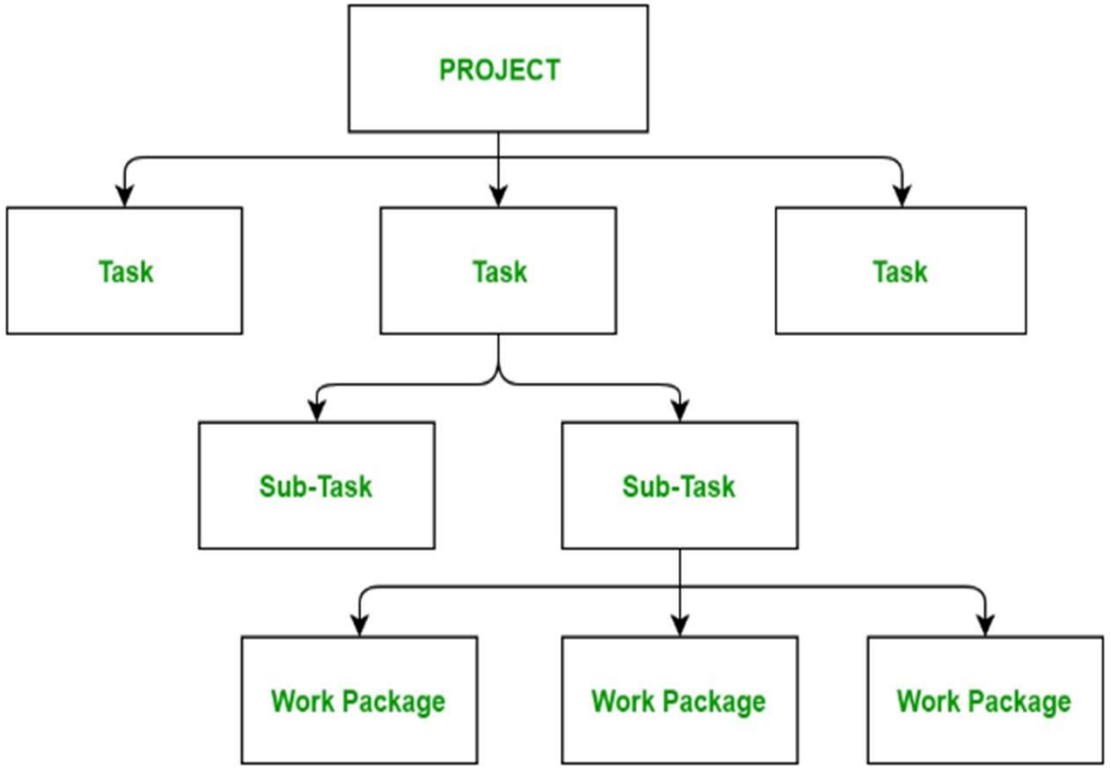
Work Breakdown Structure

- Step 1: Project managers decide project name at top / root of WBS.
- Step 2: Project managers identifies the main deliverables of the project.
- Step 3: These main deliverables are broke down into smaller higher-level tasks.
- Step 4: This complete process is done recursively to produce much smaller independent tasks.
- Step 5: Choose Task Owner. They need to get the job done.
- ➤Depends on project manager that up to which level of detail they want to break down project. ➤Lowest level tasks are most simplest & independent tasks it takes less than weeks of work.

Work Breakdown Structure

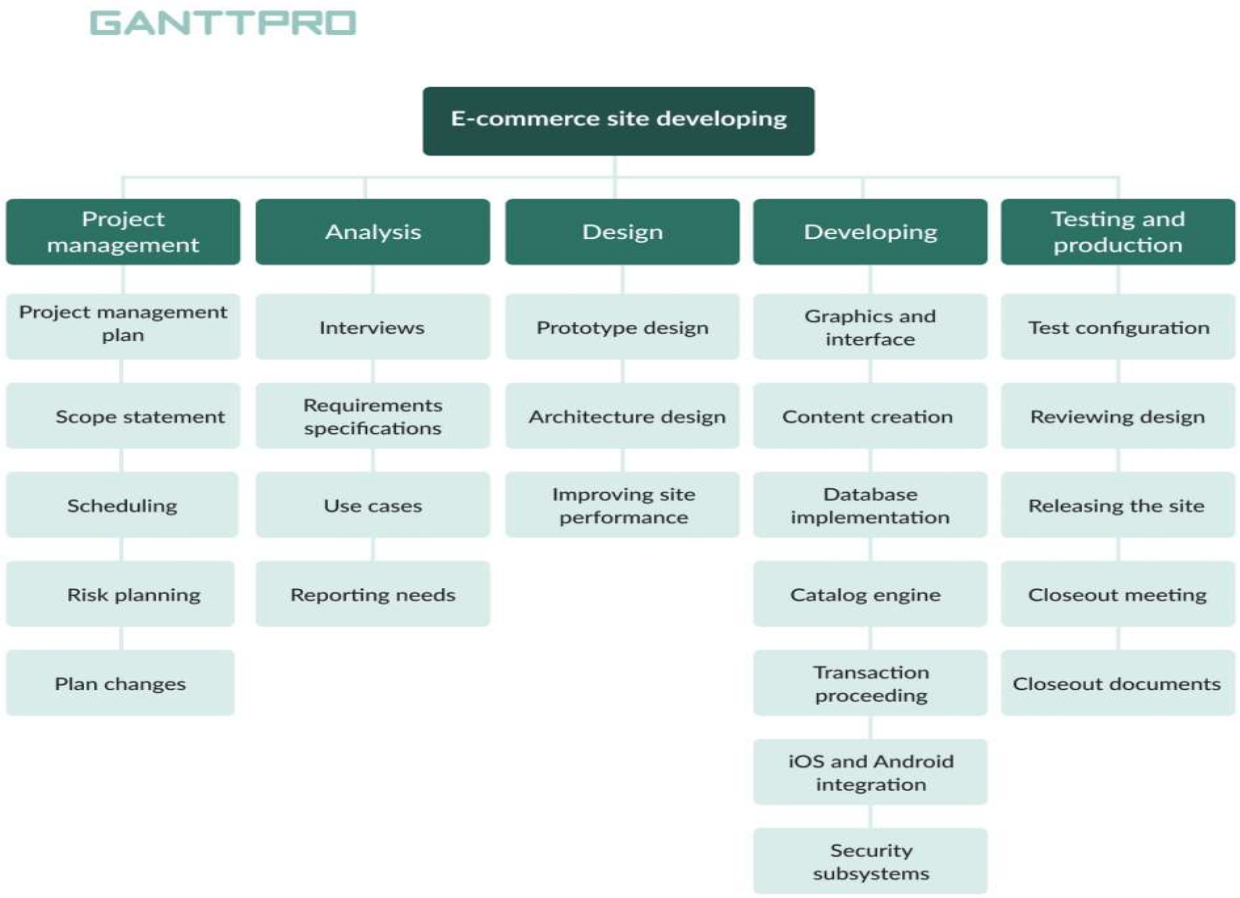
1. It allows easy management of the project.
2. It helps in proper organization of the project by the top management.
3. Giving visibility to important activities.
4. Giving visibility to risky activities.
5. Illustrate the correlation between the activities and deliverables.
6. It allows to do a precise cost estimation of each activity.
7. It allows to estimate the time that each activity will take more precisely.
8. Better communication with your team members regarding tasks
9. The efficiency of a work breakdown structure can determine the success of a project

Work Breakdown Structure



Work Breakdown Structure

Work Breakdown Structure



Scheduling methods

- Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort.
- Therefore, generalized project scheduling tools and techniques can be applied with little modification to software projects.
- Two project scheduling methods that can be applied to software development.
  1. **Program Evaluation and Review Technique (PERT)**
  2. **Critical Path Method (CPM)**

Scheduling methods

- 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
  - decomposition of the tasks that are selected

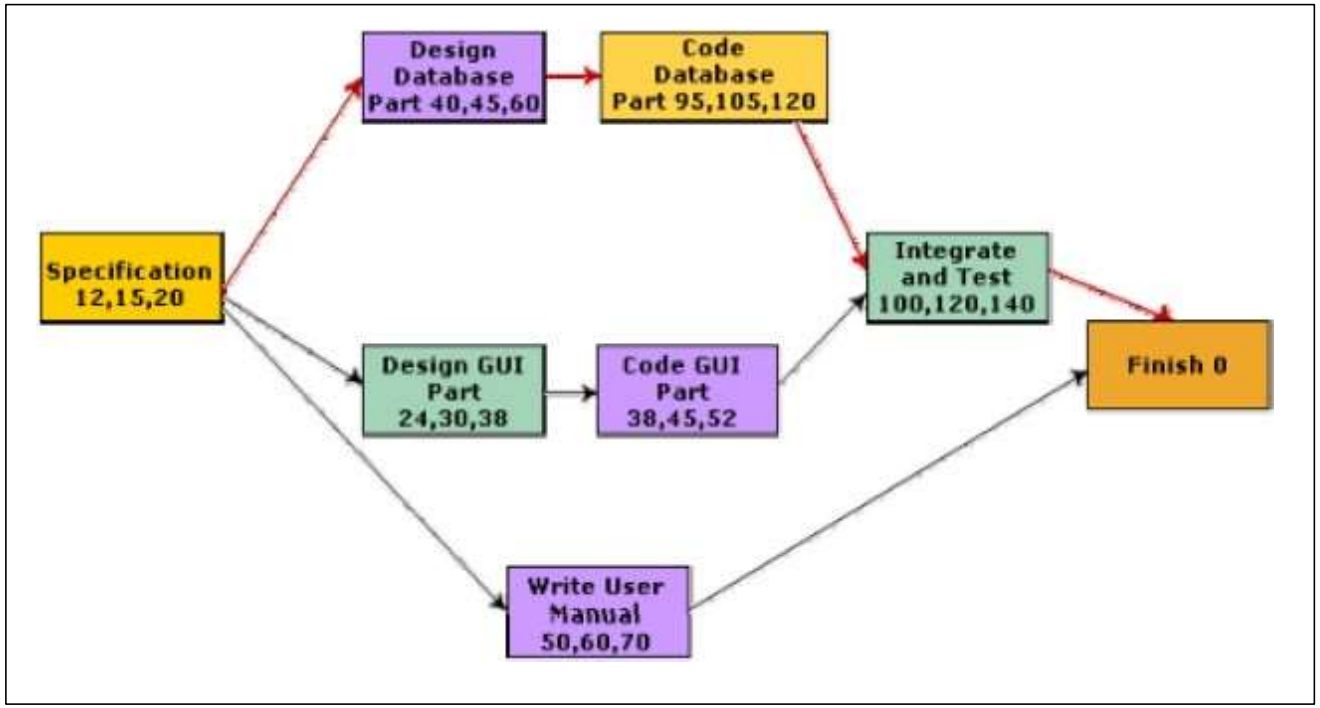
Scheduling methods

- Both PERT and CPM provide quantitative tools that allow you to:
  - **Determine the critical path**—the chain of tasks that determines the duration of the project
  - **Establish "most likely" time** estimates for individual tasks by applying statistical models
  - **Calculate "boundary times"** that define a "time window" for a particular task

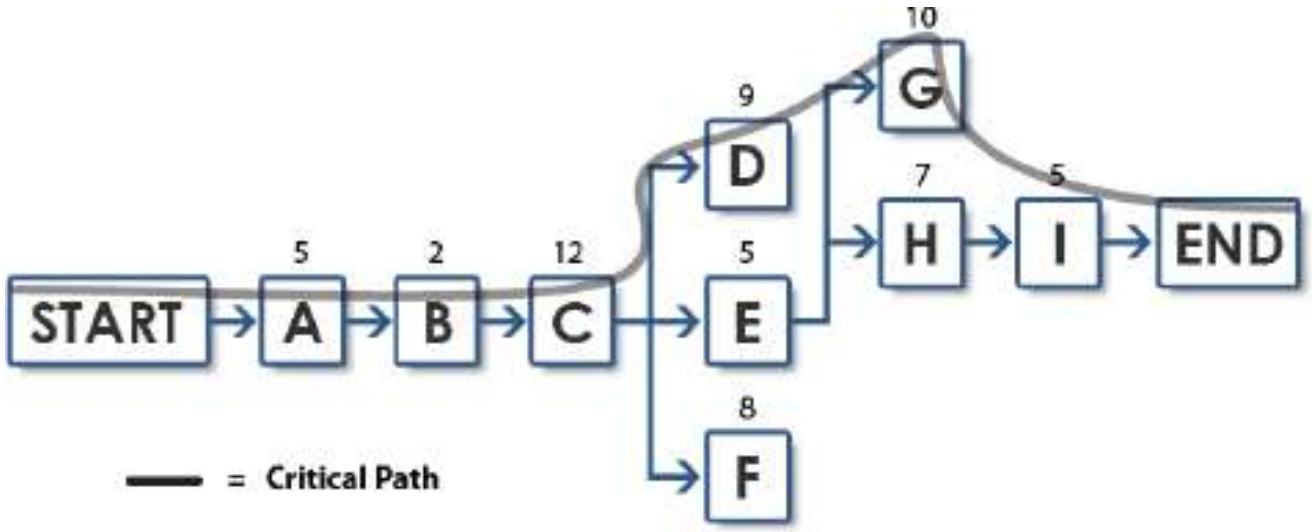
PERT Chart

- PERT (Project Evaluation and Review Technique) charts consist of a network of boxes and arrows. The boxes represent activities and the arrows represent task dependencies.
- In a PERT chart, instead of making a single estimate for each task, **pessimistic, likely and optimistic estimates** are made.
- Since all possible completion times between the minimum and maximum duration for every task has to be considered, there are not one but many critical paths.
- Gantt chart is helpful for planning the utilization of resources, while PERT chart is useful for monitoring the timely progress of activities.

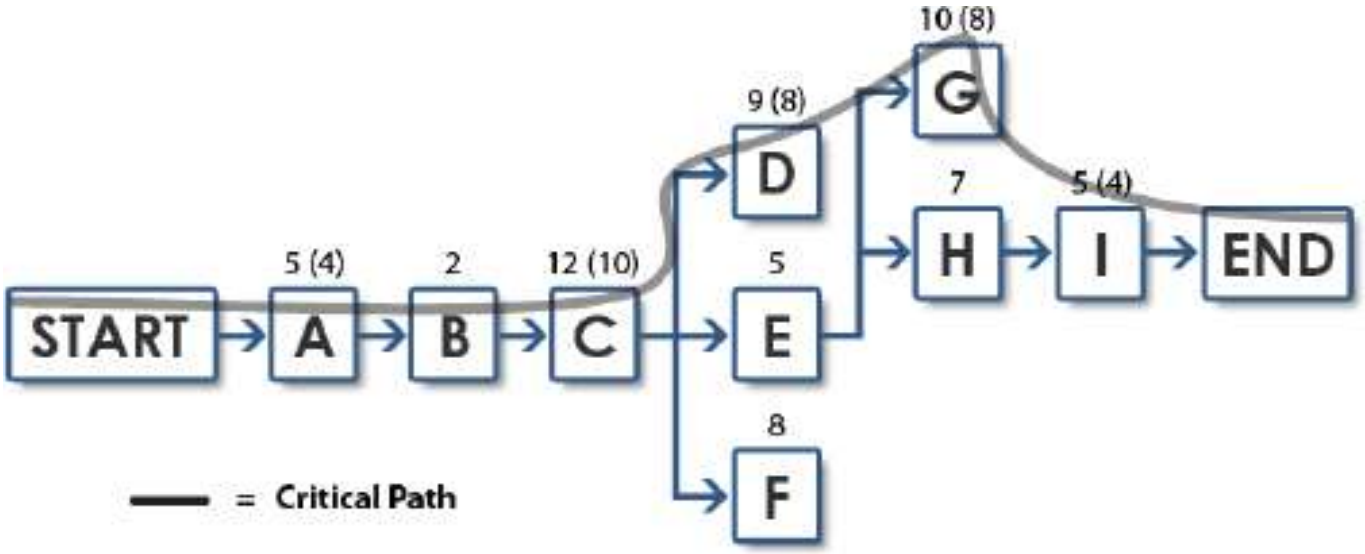
PERT Chart


Design Database Part 40,45,60
Code Database Part 95,105,120
Specification 12,15,20
Integrate and Test 100,120,140
Finish 0
Design GUI Part 24,30,38
Code GUI Part 38,45,52
Write User Manual 50,60,70

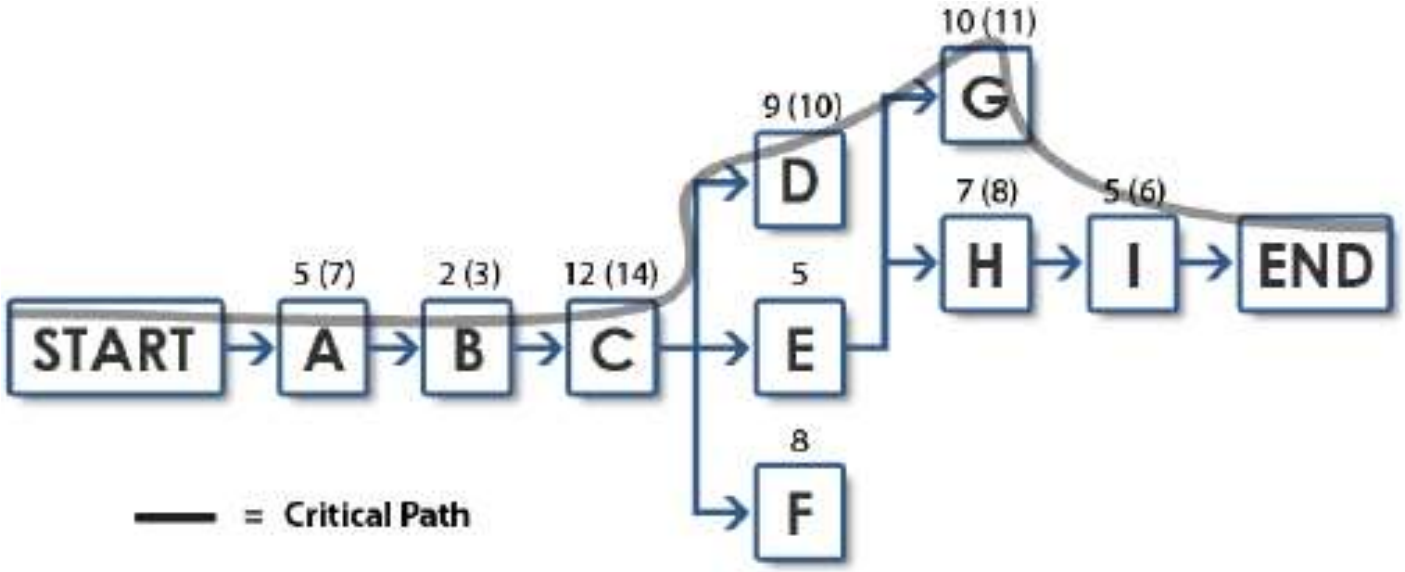
PERT Chart- Most Likely Time



PERT Chart- Optimistic Time



PERT Chart- Pessimistic Time



PERT Chart

Activity	Optimistic Time (a)	Most Likely Time (m)	Pessimistic Time (b)	Expected Time (t)	Variance
A	1	2	3	2	0.11
B	2	3	4	3	0.11
C	1	2	3	2	0.11
D	2	4	6	4	0.44
E	1	4	7	4	1
F	1	2	9	3	1.78
G	3	4	11	5	1.78
H	1	2	3	2	0.11

Expected Time (t) = (a + (4 * m) + b)/6

Variance = [(b - a)/6]^2 or (b - a)^2/36

Critical Path Method (CPM)

The Critical Path Method (CPM) is a project management technique that helps plan, schedule, and execute complex projects. It involves identifying the most important tasks and their dependencies, and then calculating how long each task will take. This information is used to determine the critical path, which is the longest sequence of tasks that must be completed to finish the project on time.

Critical Path Method (CPM)

How to find the critical path in a project:

- Step 1: Identify all tasks required to complete the project
- Step 2: Determine the sequence of tasks
- Step 3: Estimate the duration of each task
- Step 4: Draw a network diagram
- Step 5: Identify the critical path
- Step 6: Calculate the float
- Step 7: Monitor the critical path

Marwadi University
Marwadi Chandarana Group
NAAC A+

Risk Analysis and Management

Risk Analysis and Management

Risk Analysis and management are a series of steps that help a software team to understand and mange uncertainty.

Risk

- Conceptual definition of risk:
- Risk concerns future happenings.
- Risk involves change in mind, opinion, actions, places, etc.

Risk

- Risk always involves two characteristics:
- Uncertainty – the risk may or may not happen; that is, there are no 100% probable risks
- Loss –if the risk becomes a reality, unwanted consequences or losses will occur.

Types of Risk

Types of Risk

Schedule Risk:

Schedule related risks refers to time related risks or project delivery related planning risks. The wrong schedule affects the project development and delivery.

Some reasons for Schedule risks –

- Time is not estimated perfectly
- Improper resource allocation
- Frequent project scope expansion
- Failure in function identification and its' completion

Types of Risk

Budget Risk:

Budget related risks refers to the monetary risks mainly it occurs due to budget overruns. Always the financial aspect for the project should be managed as per decided but if financial aspect of project mismanaged then their budget concerns will arise by giving rise to budget risks.

Some reasons for Budget risks

- Wrong/Improper budget estimation
- Unexpected Project Scope expansion
- Mismanagement in budget handling
- Cost overruns
- Improper tracking of Budget

Types of Risk

Operational Risks :

Operational risk refers to the procedural risks means these are the risks which happen in day-to-day operational activities during project development due to improper process implementation or some external operational risks.

Types of Risk

Some reasons for Operational risks

- Insufficient resources
- Conflict between tasks and employees
- Improper management of tasks
- No proper planning about project
- Less number of skilled people
- Lack of communication and cooperation
- Lack of clarity in roles and responsibilities
- Insufficient training

Types of Risk

Technical Risks :

Technical risks refers to the functional risk or performance risk which means this technical risk mainly associated with functionality of product or performance part of the software product.

Some reasons for Technical risks –

- Frequent changes in requirement
- Less use of future technologies
- Less number of skilled employee
- High complexity in implementation
- Improper integration of modules

Types of Risk

Programmatic Risks :

Programmatic risks refers to the external risk or other unavoidable risks. These are the external risks which are unavoidable in nature. These risks come from outside and it is out of control of programs.

Some reasons for Programmatic risks –

- Rapid development of market
- Running out of fund / Limited fund for project development
- Changes in Government rules/policy
- Loss of contracts due to any reason

Types of Risk

Project risks:

- It threaten the project plan.
- That is, if project risks become real, it is likely that project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, customer, and requirements problems and their impact on a software project.
- Project complexity, size, and the degree of structural uncertainty were also defined as project (and estimation) risk factors

Types of Risk

Technical risks:

- It threaten the quality and timeliness of the software to be produced.
- If a technical risk becomes a reality, implementation may become difficult or impossible.
- Technical risks identify potential design, implementation, interface, verification, and maintenance problems.
- In addition, specification ambiguity, technical uncertainty, and "leading-edge" technology are also risk factors.
- Technical risks occur because the problem is harder to solve than we thought it would be.

Types of Risk

Business risks:

It threaten the viability of the software to be built. Business risks often jeopardize the project or the product. Candidates for the top five business risks are

- building an excellent product or system that no one really wants (market risk)
- building a product that no longer fits into the overall business strategy for the company (strategic risk)

Types of Risk

- building a product that the sales force doesn't understand how to sell
- losing the support of senior management due to a change in focus or a change in people (management risk)
- losing budgetary or personnel commitment (budget risks). It is extremely important to note that simple categorization won't always work. Some risks are simply unpredictable in advance.

Another general category of risks

1. **Known risks**
   - Those risks 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 (Ex. unrealistic delivery date, lack of documentation requirements or software scope, poor development environment)
2. **Predictable risks**
   - Those risks that are deduced from past project experience(e.g., staff turnover, poor communication with the customer, etc).
3. **Unpredictable risks**
   - Those risks that can and do occur, but are extremely difficult to identify in advance.

Risk Strategies

1. **Reactive Risk Strategies**
2. **Proactive Risk Strategies**

Reactive Risk Strategies

Reactive Risk Strategy:

- A reactive approach deals with risks after they have occurred. This strategy focuses on damage control, recovery, and contingency plans.

Key Characteristics:

- Waits for risks to materialize before taking action.
- Focuses on crisis management and resolution.
- Involves contingency plans and corrective measures.
- Used when risks are unpredictable or difficult to anticipate.

Proactive Risk Strategies

Proactive Risk Strategy:

- A proactive approach focuses on identifying and mitigating risks before they become problems. It emphasizes prevention and risk assessment.

Key Characteristics:

- Identifies potential risks early through risk assessment and analysis.
- Implements preventive measures to avoid risks.
- Uses predictive analytics and monitoring systems.
- Encourages a culture of preparedness and continuous improvement.

Steps for Risk Management

- **Risk identification:** Identify possible risks and recognize what can go wrong.
- **Risk Estimation(Projection):** Analyze each risk to estimate the probability that it will occur and the damage that it will do if it will occur.
- **Risk Mitigation & Planning:** Rank the risks by probability and impact
- Impact may be negligible, marginal, critical, and catastrophic.
- Develop a contingency plan to manage those risks having high probability and high impact.

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.
- Two distinct types of risks:
- **Generic risks** are a potential threat to every software project.
- **Product-specific risks** can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the project at hand

Risk Item Checklist

- One method for identifying risks is to create a **risk item checklist**.
- The checklist can be used for risk identification and focuses on some subset of **known and predictable risks in the following generic subcategories**:
  - **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.
  - **Customer characteristics** —risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.

Risk Item Checklist

- **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 Projection (Estimation)

- Risk projection, also called risk estimation, attempts to rate each risk in two ways—the
  - Likelihood or probability that the risk is real
  - The consequences(result) of the problems associated with the risk, should it occur.
- The project planner, along with other managers and technical staff, performs four risk projection activities:
  1. Establish a scale that reflects the perceived likelihood of a risk Ex., 1-low, 10-high
  2. Explain the consequences of the risk
  3. Estimate the impact of the risk on the project and the product
  4. Assess the overall accuracy of the risk projection so that there will be no misunderstandings.

Developing RiskTable

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	
•				
•				
•				

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

PS-Product Size
BU-Business Impact
CU-Customer Characteristic
PR-Process definition
TE-Technology
DE-Development Environment
ST-Staff size and experience

Developing RiskTable

- Once table is completed, manger will give order of prioritization to the risk. Therefore, the table is sorted by probability and by impact.
- High-probability, high-impact risks get into the top of the table, and low-probability risks drop to the bottom. (First order prioritization).
- The project manager studies the resultant sorted table and defines a cutoff line.
- The cutoff line implies that only risks that lie above the line will be given further attention.
- Risks that fall below the line are considered as second-order prioritization.

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 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 represent the risk is Condition-Transition-Consequence(CTC) format.
- Given that <condition>then there is concern that <consequences>.

RMMM(Risk Mitigation, Monitoring, and Management)

- An effective strategy for dealing with risk must consider three issues
  - Risk mitigation (i.e., avoidance)
  - Risk monitoring(Tracking)
  - Risk management and contingency planning

Risk Mitigation (avoidance)

- If a software team adopts a proactive approach to risk, avoidance is always the best strategy. This is achieved by developing a plan for risk mitigation.
- For Ex., **Risk of high staff turnover(Replacement)**
  - To mitigate this risk, project management would develop a strategy for reducing turnover.
  - The possible steps to be taken are:

1. Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, and competitive job market).
2. Mitigate those causes that are under project management's control before the project starts.

Risk Mitigation (avoidance)

3. Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
4. Organize project teams so that information about each development activity is widely dispersed.
5. Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner.
6. Assign a backup staff member for every critical technologist.

Risk Monitoring

- The project manager monitors factors that may provide an indication whether the risk is becoming more or less likely
- In the case of high staff turnover, project manager monitors
  - attitude of team members based on project pressure
  - Inter-personal relationship among team members
  - Availability of jobs outside the company
  - In addition to monitoring these factors, a project manager should monitor the effectiveness of risk mitigation steps.

Risk management and contingency planning

- Risk management assumes that mitigation efforts have failed and that the risk has become the reality.
- If the mitigation strategy is followed then backup is available, information is documented, and knowledge has been dispersed across the team.
- In addition, project manager can temporarily refocus resources and readjust the project schedule to those functions which are fully staffed. This step will enable "New comers" in the team to "get up to the speed".
- Those 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," and/or meeting with other team members who will remain on the project.

RMMM PLAN

- The RMMM PLAN documents all work performed as part of risk analysis and 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, RIS is maintained using a database system.
- So Creation and information entry, priority ordering, searches and other analysis may be accomplished easily.
- Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence.

Risk information sheet (RIS)

Risk information sheet			
Risk ID: P02-4-32	Date: 5/9/02	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 7/1/02

Current status:
5/12/02: Mitigation steps initiated.

Originator: D. Gagne	Assigned: B. Laster

Safety Risks and Hazards

- Safety is a property of a system that reflects the system's ability to operate, normally or abnormally, without danger of causing human injury or death and without damage to the system's environment.
- It is important to consider software safety as most devices whose failure is critical now incorporate software-based control systems.
- Safety and reliability are related but distinct. Reliability is concerned with conformance to a given specification and delivery of service. Safety is concerned with ensuring system cannot cause damage irrespective of whether or not it conforms to its specification.
- System reliability is essential for safety but is not enough

THANK YOU