

UNIT 3

Managing Software Project

Prepared by:
Dr. Pooja M Bhatt
Computer Engineering Department,
MBIT

Index

- Project Management Spectrum
- W5HH Principal
- Software Metrics(Process, Product and Project Metrics)
- Software Project Estimations
- Project Scheduling and Tracking
- Risk Analysis and Management
- Risk Identification
- Risk Projection
- Risk Refinement
- Risk Mitigation

Why Measure Software?

- To **determine** (to define) **quality** of a product or process.
- To **predict qualities** of a product or process.
- To **improve quality** of a product or process.

Terminologies

- **Measure**
- It provides a **quantitative indication** of the amount, dimension, capacity or size of some product or process
- Ex. the number of uncovered errors
- **Metrics**
- It is a **quantitative measure** of the degree system, component or process possesses attribute
- It **relates individual measures** in some way
- Ex. number of errors found per review
- **Direct Metrics :**
- Direct measure of the software process & Product
- E.g. Lines of code (LOC), execution speed, and defect

Terminologies

- **Indirect Metrics**
- **Aspects that are not immediately quantifiable**
- Ex. **Functionality, Quantity, Reliability**
- **Indicators**
- It is 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 (analysis and specification metrics)** is an indicator of increased coding, integration and testing effort
- **Faults**
- **Errors** - Faults found by the practitioners during software development
- **Defects** - Faults found by the customers after release

Project management concerns

Manager concerns about following issues:

- Product quality
- Risk Assessment
- Measurement
- Cost Estimation
- Project Schedule
- Customer Communication
- Staffing
- Other Resources
- Project Monitoring

Why Project Fail?

- Changing customer requirement
- Ambiguous/Incomplete requirement
- Unrealistic deadline
- An honest underestimate of effort
- Predictable and/or unpredictable risks
- Technical difficulties
- Miscommunication among project staff

Management Spectrum

- Effective project management focuses on four aspects of the project known as the 4 P's:
- **People** - The most important element of a successful project. (recruiting, selection, performance management, training, compensation, career development, organization, work design, team/culture development)
- **Product** - The software to be built (product objectives, scope, alternative solutions, constraint)
- **Process** - The set of framework activities and software engineering tasks to get the job done (framework activities populated with tasks, milestones, work products, and QA points)
- **Project** - All work required to make the product a reality. (planning, monitoring, controlling)

People

Player of the project:

- The Stakeholders
- Team leaders
- The Software Team
- Agile Team (Implementer)
- Coordination and Communication Issues

Stakeholders

- **Senior managers** who define the business issues that often have significant influence on the project.
- **Project (technical) managers** who must plan, motivate, organize, and control the practitioners who do software work.
- **Practitioners** who deliver the technical skills that are necessary to engineer a product or application.
- **Customers** who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
- **End-users** who interact with the software once it is released for production use.

Team Leaders

- **MOI model** for leadership
- **Motivation:** The ability to encourage (by “push or pull”) technical people to produce to their best ability.
- **Organization:** The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
- **Ideas or Innovation:** The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.
- Characteristics of effective project managers (problem solving, managerial identity, achievement, influence and team building)

Software Teams

How to lead?

How to organize?

How to collaborate?



How to motivate?

How to create good ideas?

Software Teams

- **The following factors must be considered when selecting a software project team structure ...**
- The difficulty of the problem to be solved
- The size of the resultant program(s) in lines of code or function points
- The time that the team will stay together (team lifetime)
- The degree to which the problem can be modularized
- The required quality and reliability of the system to be built
- The rigidity of the delivery date
- The degree of sociability (communication) required for the project

Organizational Paradigms

- **Closed paradigm**— structures a team along a traditional hierarchy of authority. Less likely to be innovative when working within the closed paradigm.
- **Random paradigm**— structures a team loosely and depends on individual initiative of the team members. It struggles when “orderly performance” is required.
- **Open paradigm**— attempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm
- **Synchronous paradigm**— relies on the natural compartmentalization of a problem and organizes team members to work on pieces of the problem with little active communication among themselves

Agile Team

- Small, Highly motivated project team also called Agile Team, adopts many of the characteristics of successful software projects.
- Team members must have trust in one another.
- The distribution of skills must be appropriate to the problem.
- Unconventional person may have to be excluded from the team, if team organized is to be maintained.
- Team is “self-organizing”
- An adaptive team structure
- Uses elements of organizational paradigm’s random, open, and synchronous paradigms
- Significant autonomy

Team Coordination & Communication

- **Formal, impersonal approaches**
- include software engineering documents and work products (including source code), technical memos, project milestones, schedules, and project control tools, change requests and related documentation, error tracking reports, and repository data.
- **Formal, interpersonal procedures**
- focus on quality assurance activities applied to software engineering work products. These include **status review meetings and design and code inspections.**
- **Informal, interpersonal procedures**
- include group meetings for information dissemination and problem solving and “collocation of requirements and development staff.”
- **Electronic communication**
- encompasses electronic mail, electronic bulletin boards, and by extension, video-based conferencing systems.
- **Interpersonal networking**
- includes informal discussions with team members and those outside the project who may have experience or insight that can assist team members

Product Scope

- Software Scope:
- Context: How does the software to be built fit into a larger system, product, or business context and what constraints are imposed as a result of the context?
- Information objectives :What customer-visible data objects are produced as output from the software? What data objects are required for input?
- Function and performance: What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?
- Software project scope must be unambiguous and understandable at the management and technical levels.

Problem Decomposition

- Sometimes called partitioning or problem elaboration
- Decomposition is applied in **2 major areas**
- Functionality that must be delivered.
- Process that will be used to deliver it.
- Once scope is defined ...
- It is decomposed into constituent functions
- It is decomposed into user-visible data objects

or

- It is decomposed into a set of problem classes
- Decomposition process continues until all functions or problem classes have been defined
- Decomposition will make planning easier.

The Process

- Process model chosen must be appropriate for the:
 - Customers and developers,
 - Characteristics of the product, and
 - Project development environment
- Once a process framework has been established
 - Consider project characteristics
 - Determine the degree of thoroughness required
 - Define a task set for each software engineering activity
 - Task set =
 - Software engineering tasks
 - Work products
 - Quality assurance points
 - Milestones

Melding the product and process

Common process framework activities	Customer communication					Planning					Risk analysis					Engineering				
Software engineering tasks																				
Product functions																				
Text input																				
Editing and formatting																				
Automatic copy edit																				
Page layout capability																				
Automatic indexing and TOC																				
File management																				
Document production																				

Melding the Product and Process

- Project planning begins with melding the product and the process
- Each function to be engineered must pass through the set of framework activities defined for a software organization
- The job of the project manager is to estimate the resources required to move each function through the framework activities to produce each work product

Process decomposition

Process decomposition begins when the project manager tries to determine how to accomplish each activity.

E.g. A small, relatively simple project might require the following work tasks for the communication activity:

1. Develop list of clarification issues.
2. Meet the customer to address clarification issues.
3. Jointly develop a statement of scope.
4. Review the statement of scope with all concerned.
5. Modify the statement of scope as required.

The Project

- Projects get into jeopardy(failure) when ...
 - Software people don't understand their customer's needs.
 - The product scope is poorly defined.
 - Changes are managed poorly.
 - The chosen technology changes.
 - Business needs change [or are ill-defined].
 - Deadlines are unrealistic.
 - Users are resistant.
 - Sponsorship is lost [or was never properly obtained].
 - The project team lacks people with appropriate skills.
 - Managers [and practitioners] avoid best practices and lessons learned.

Common-Sense Approach

- ***Start on the right foot.*** This is accomplished by working hard (very hard) to understand the problem that is to be solved and then setting realistic objectives and expectations.
- ***Maintain momentum.*** The project manager must provide incentives to keep turnover of personnel to an absolute minimum, the team should emphasize quality in every task it performs, and senior management should do everything possible to stay out of the team's way.
- ***Track progress.*** For a software project, progress is tracked as work products (e.g., models, source code, sets of test cases) are produced and approved (using formal technical reviews) as part of a quality assurance activity.
- ***Make smart decisions.*** In essence, the decisions of the project manager and the software team should be to "keep it simple."
- ***Conduct a postmortem analysis.*** Establish a consistent mechanism for extracting lessons learned for each project. Evaluate plan, schedule, analysis of project, customer feedback, etc in written form.

To Get to the Essence of a Project - W⁵HH Approach

- Boehm suggests an approach(W⁵HH) that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches, and required resources.
- *It applicable regardless of size or complexity of software project*

1. Why is the system being developed?

Enables all parties to assess the validity of business reasons for the software work

2. What will be done?

Establish the task set that will be required.

3. When will it be accomplished?

Project schedule to achieve milestone.

4. Who is responsible?

Role and responsibility of each member.

5. Where are they organizationally located?

Customer, end user and other stakeholders also have responsibility.

6. How will the job be done technically and managerially?

Management and technical strategy must be define.

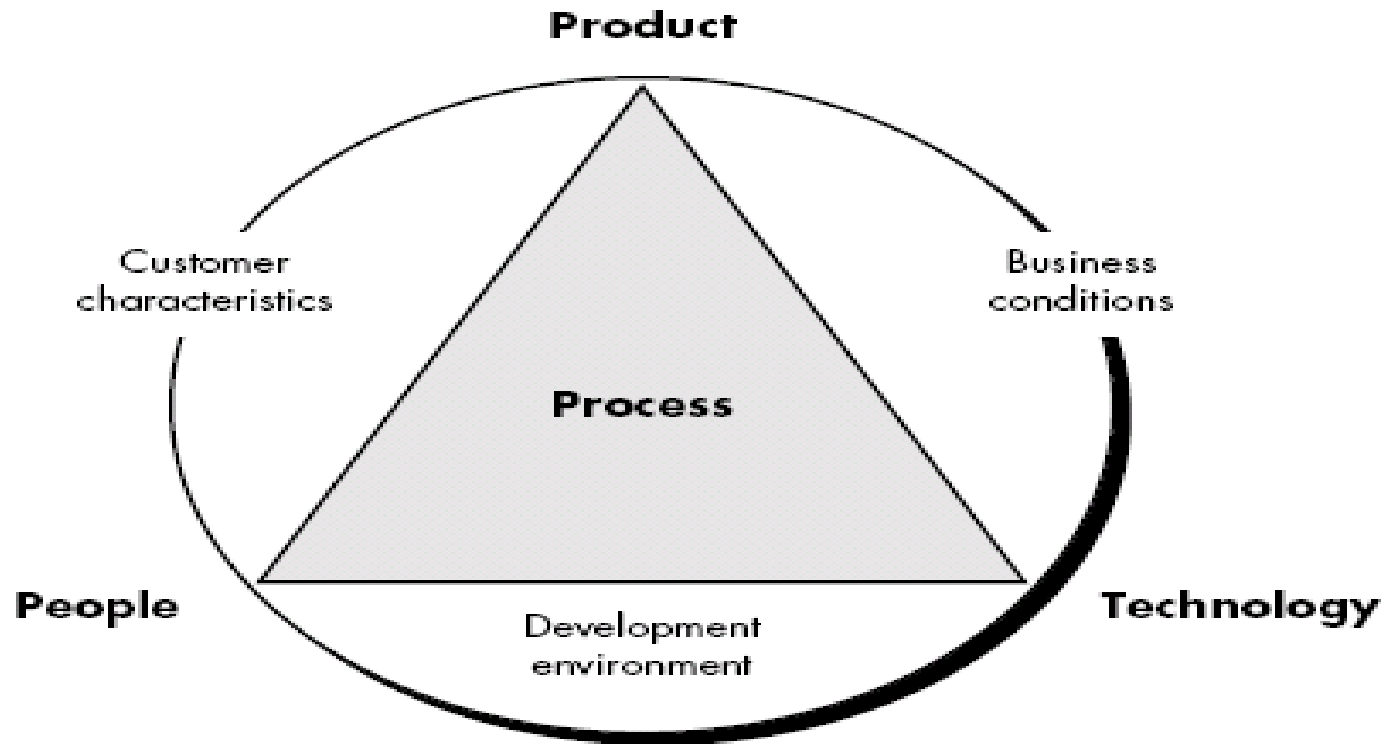
7. How much of each resource is needed?

Develop estimation.

Process, project and measurement

- Process Metrics:-
 - They Are collected across all projects and over long periods of time. Their intent is to provide a set of process indicator that lead to long term software process improvement.
- Project Metrics:-
 - They enables a software project manager to
 1. Assess the status of an ongoing project
 2. Track potential risks.
 3. Uncover problem areas before they go “Critical”
 4. Adjust work flow or tasks
 5. Evaluate the project team’s ability to control quality of software work products.
- Measurement :-
 - They Are collected by a project team and converted into process metrics during software process improvement.

Process Metrics and Software Process Improvement



Process Metrics and Software Process Improvement

- Process at the center connecting 3 factors that have a profound influence on software quality and organizational performance.
- Process triangle exists within a circle of environmental conditions that include the development environment, business conditions and customer characteristics.
- We measure the efficacy of a software process indirectly.
 - That is, we derive a set of metrics based on the outcomes that can be derived from the process.
 - Outcomes include
 - measures of errors uncovered before release of the software
 - defects delivered to and reported by end-users
 - work products delivered (productivity)
 - human effort expended
 - calendar time expended
 - schedule conformance
 - other measures.
- We also derive process metrics by measuring the characteristics of specific software engineering tasks.

Process Metrics Guidelines

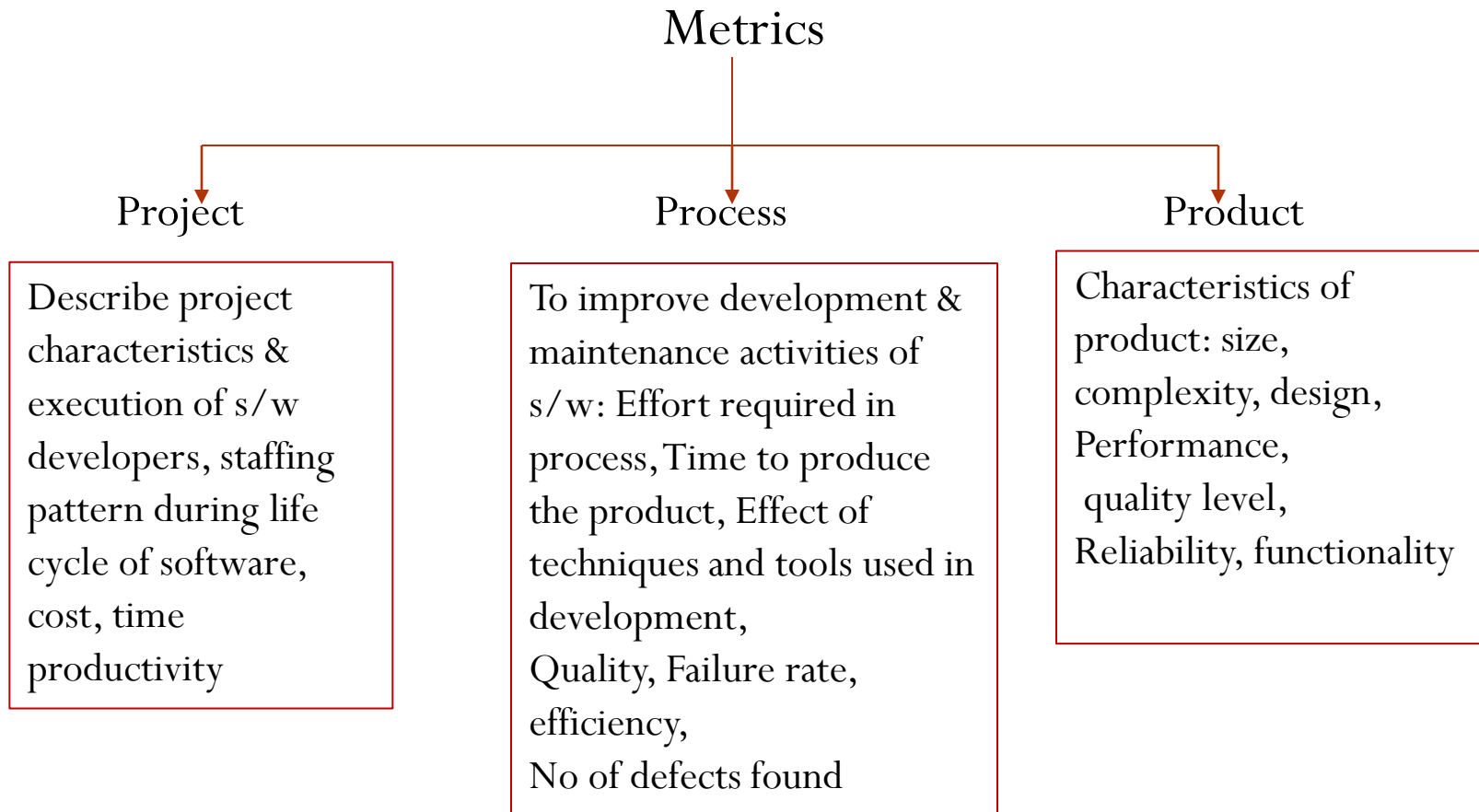
- Use common sense and organizational sensitivity when interpreting metrics data.
- Provide regular feedback to the individuals and teams who collect measures and metrics.
- Don't use metrics to appraise individuals.
- Work with practitioners and teams to set clear goals and metrics that will be used to achieve them.
- Never use metrics to threaten individuals or teams.
- Metrics data that indicate a problem area should not be considered "negative." These data are merely an indicator for process improvement.
- Don't obsess on a single metric to the exclusion of other important metrics.

Project Metrics

- Used to minimize the development schedule by making the adjustments necessary to avoid delays and mitigate potential problems and risks
- Used to assess product quality on an ongoing basis and, when necessary, modify the technical approach to improve quality.
- Every project should measure:
 - *Inputs*—measures of the resources (e.g., people, tools) required to do the work.
 - *Outputs*—measures of the deliverables or work products created during the software engineering process.
 - *Results*—measures that indicate the effectiveness of the deliverables.

Metrics

- Software Metric is standard of measure that contains many activities which involve some degree of measurement.



Software Measurement

Categories in 2 ways:

- ***Direct measure*** of the software process & Product
 - E.g. Lines of code (LOC), execution speed, and defect)
- ***Indirect measures*** of the product that include functionality, complexity, efficiency, reliability, maintainability etc.

Types of Metrics for measuring software size

- SIZE oriented
- Function oriented
- Object Oriented
- Use case oriented

Size-Oriented metrics

Size-oriented metrics measures on LOC as normalization value.

- Errors per KLOC (thousand lines of code)
- Defects per KLOC
- \$ per LOC
- Pages of documentation per KLOC
- Errors per person-month
- Errors per review hour
- LOC per person-month
- \$ per page of documentation

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
•	•	•	•	•	•		
•	•	•	•	•	•		
•	•	•	•	•	•		

analysis, design
,code and test

Before Release

After Release

Lines of Code

- Don't count comment and blank line for size estimation as they do not contribute to any kind of functionality or they can be misused by developer to give false notions about productivity.
- **Advantage:** Easy to count and calculate from code
- **Disadvantage:** language dependent, technology dependent and LOC count techniques varies in diff. organization

Example for LOC

```
1. #include<stdio.h>
2. int main()
3. {
4. int a;
5. int b;
6. int sum;
7. a=10;
8. b=20;
9. sum=0;
10. sum=a+b;
11. printf("\nSum=");
12. printf("%d",sum);
13. return 0;
14. }
```



```
1. #include<stdio.h>
2. void main()
3. {
4. int a=10,b=20,sum=0;
5. sum=a+b;
6. printf("\nSum=%d",sum);
7. }
```

Function-Oriented Metrics

- It use a measure of functionality delivered by the application as a normalization value.
- Since 'functionality' cannot be measured directly, it must be derived indirectly using other direct measures
- **Function Point (FP)** is widely used as function oriented metrics.
- FP derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.
- FP is based on characteristic of Software information domain and complexity.
- Like LOC measure, FP is controversial.
- FP is programming language independent.
- It is ideal for applications using conventional and nonprocedural languages.

FP- Five information domain characteristics

Measurement parameter (functional units)	Weighting factor					
	Simple		Average		Complex	
Number of user inputs	3 x _		4 x _		6 x _	
Number of user outputs	4 x _		5 x _		7 x _	
Number of user inquiries	3 x _		4 x _		6 x _	
Number of internal logical files	7 x _		10 x _		15 x _	
Number of external interfaces	5 x _		7 x _		10 x _	
Count Total	Simple Total		Average Total		Complex total	

FP- Five information domain characteristics

- **Number of user inputs** - Each user input that provides distinct data to the software is counted
- **Number of user outputs** - Each user output that provides information to the user is counted. Output refers to reports, screens, error messages, etc
- **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, google search)
- **Number of files** - Each logical master file (i.e. large database or separate file) is counted.
- **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.

FP- Five information domain characteristics

- **Value Adjustment Factors**

- F1. Data Communication
- F2. Distributed Data Processing
- F3. Performance
- F4. Heavily Used Configuration
- F5. Transaction Role
- F6. Online Data Entry
- F7. End-User Efficiency
- F8. Online Update
- F9. Complex Processing
- F10. Reusability
- F11. Installation Ease
- F12. Operational Ease
- F13. Multiple Sites
- F14. Facilitate Change

FP- Five information domain characteristics

- To compute function points (FP), the following relationship is used:

$$FP = count\ total\ [0.65 + 0.01 \sum(Fi)]$$

The F_i ($i = 1$ to 14) are "complexity adjustment values".

- 14 questions assign of on scale of 0 to 5
(no influences, Incidental, Moderate, Significant and essential)
- **Each of these values measure on scale based ranges from 0 (not important or applicable) to 5 (absolutely essential)**
- **Count total is unadjusted function point.**
- Once function points have been calculated, they are used in a manner analogous to LOC as a way to normalize measures for software productivity, quality, and other attributes:
 - Errors per FP.
 - Defects per FP.
 - \$ per FP.
 - Pages of documentation per FP.
 - FP per person-month.

Example 1

- Compute the Function Point (FP) value for a project with the following details.

Information domain characteristics	Average Weighing Factor
Number of user inputs: 32	4
Number of user outputs: 60	5
Number of user inquiries: 24	4
Number of files: 8	10
Number of external interfaces: 2	7

- Note: Assume that all complexity adjustment values are average. Assume that 14 algorithms have been counted. So the value adjustment factor is $(\sum Fi)$ is 35.

Solution

Parameter	Count	Average	Product
User Inputs	32	4	128 (32 * 4)
User Outputs	60	5	300 (60 * 5)
Inquiries	24	4	96 (24 * 4)
Files	8	10	80 (8 * 10)
External Interfaces	2	7	14 (2 * 7)
Count Total			618

Solution

- Here, Complexity Adjustment Values are average so we multiply 14 by 2.5. because the minimum value for Complexity Adjustment Values is 0 and maximum value for Complexity Adjustment Value is 5. So we take average as
- $(0 + 5) / 2.$
- So, for us $\sum F_i = 14 * 2.5 = 35$ (complexity adjustment values)
- $FP = \text{count total} * [0.65 + 0.01 * \sum F_i]$
- $= 618 * [0.65 + 0.01 * 35]$
- $= 618 * [0.65 + 0.35]$
- $= 618 * [1.00]$
- **FP= 618**

Example 2

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

Used Adjustment Factors and assumed values are,

F09. Complex internal processing = **3**

F10. Code to be reusable = **2**

F13. Multiple sites = **3**

F03. High performance = **4**

F02. Distributed processing = **5**

Project Adjustment Factor (VAF) = 17

$$FP = \text{Count Total} * [0.65 + 0.01 * \sum(F_i)]$$

$$FP = [50] * [0.65 + 0.01 * 17]$$

$$FP = [50] * [0.65 + 0.17]$$

$$FP = [50] * [0.82] = 41$$

Example 3

Q: Study of requirement specification for project has produced following results: Need of 7 inputs,10 outputs,6 inquiries,17 files,4 external interfaces.

Note:

Input and output interface function point attributes are average complexity. All other function points attributes are of low complexity. Find adjustment function points and assuming complexity value is 32.

Information Domain Value	Count		Weighting factor				
			Simple	Average	Complex		
External Inputs (EIs)	7	×	3	4	6	=	28
External Outputs (EOs)	10	×	4	5	7	=	40
External Inquiries (EQs)	6	×	3	4	6	=	18
Internal Logical Files (ILFs)	17	×	7	10	15	=	119
External Interface Files (EIFs)	4	×	5	7	10	=	28
Count total							233

Value adjustment factors (VAF) = 32 given

$$\begin{aligned}
 FP &= \text{Count Total} * [0.65 + 0.01 * \Sigma(F_i)] \\
 &= 233 * [0.65 + 0.01 * 32] \\
 &= 233 * 0.97 = 226.01
 \end{aligned}$$

OBJECT-ORIENTED METRICS

- Primary objectives for object-oriented metrics are no different than those for metrics derived for conventional software:
 - To better understand the quality of the product
 - To assess the effectiveness of the process
 - To improve the quality of work performed at a project level

CHARACTERISTICS OF OBJECT-ORIENTED METRICS

- Metrics for OO systems must be tuned to the characteristics that distinguish OO from conventional software.
- So there are five characteristics that lead to specialized metrics:
 - Localization
 - Encapsulation
 - Information hiding,
 - Inheritance, and
 - Object abstraction techniques.

Localization

- *Localization* is a characteristic of software that indicates the manner in which information is concentrated within a program.
- For example, in conventional methods for *functional decomposition* localize information around *functions* & *Data-driven* methods localize information around specific *data structures*.
- But In the OO context, information is concentrated by summarize both *data and process* within the bounds of a *class or object*.
- Since the class is the basic unit of an OO system, localization is based on objects.
- Therefore, metrics should apply to the class (object) as a complete entity.
- Relationship between operations (functions) and classes is not necessarily one to one.
- Therefore, classes collaborate must be capable of accommodating one-to-many and many-to-one relationships.

Encapsulation

- Defines encapsulation as “the packaging (or binding together) of a collection of items
- For conventional software,
 - Low-level examples of encapsulation include records and arrays,
 - mid-level mechanisms for encapsulation include functions, subroutines, and paragraphs
- For OO systems,
 - Encapsulation include the responsibilities of a class, including its *attributes and operations*, and the states of the class, as defined by specific attribute values.
- Encapsulation influences metrics by changing the focus of measurement from a *single module to a package of data (attributes) and processing modules (operations)*.

Information Hiding

- Information hiding suppresses (or hides) the operational details of a program component.
- Only the information necessary to access the component is provided to those other components that wish to access it.
- A well-designed OO system should encourage information hiding. And its indication of the quality of the OO design.

Inheritance

- Inheritance is a mechanism that enables the responsibilities of one object to be propagated to other objects.
- Inheritance occurs throughout all levels of a class hierarchy. In general, conventional software does not support this characteristic.
- Because inheritance is a crucial characteristic in many OO systems, many OO metrics focus on it.

Abstraction

- Abstraction focus on the essential details of a program component (either data or process) with little concern for lower-level details.
- Abstraction is a relative concept. As we move to higher levels of abstraction we ignore more and more details.
- Because a class is an abstraction that can be viewed at many *different levels of detail* and in a *number of different ways* (e.g., as a list of operations, as a sequence of states, as a series of collaborations), OO metrics represent abstractions in terms of *measures of a class*

OBJECT ORIENTED METRICS

- NO OF SCENARIO PER SCRIPTS

Eg:(senario,action,scripts)

- NO OF KEY CLASSES
- NO OF SUPPORT CLASSES
- AVERAGE NO OF SUPPORT CLASSES PER KEY CLASSES
- NO OF SUBSYSTEM

CK metrics suite

- CK have proposed six class-based design metrics for OO systems.

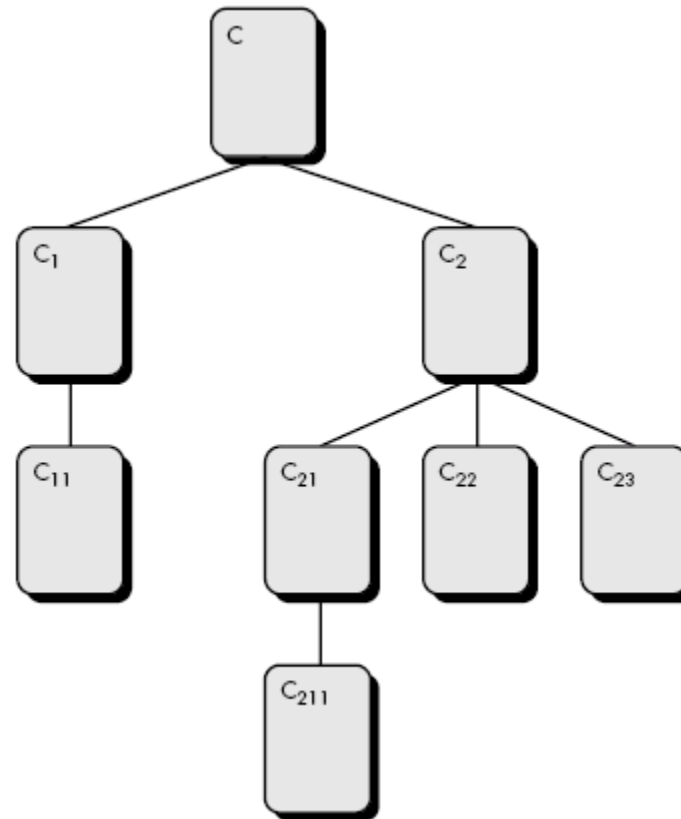
1. Weighted methods per class (WMC):-

- Assume that n methods of complexity c_1, c_2, \dots, c_n are defined for a class **C**.
- The specific complexity metric that is chosen (e.g., cyclomatic complexity) should be normalized so that nominal complexity for a method takes on a value of 1.0.

$$WMC = \sum c_i$$

- for $i = 1$ to n . The number of methods and their complexity are reasonable indicators of the amount of effort required to implement and test a class.
- So if no. of methods are increase, complexity of class also increase. Therefore, limiting potential reuse (i.e. use inheritance concept)

2. Depth of the inheritance tree (DIT):-



- This metric is “the maximum length from the node to the root of the tree”
- Referring to Figure, the value of DIT for the class-hierarchy shown is 4.
- As DIT grows, it is likely that lower-level classes will inherit many methods. **This leads to potential difficulties when attempting to predict the behavior of a class.**
- A deep class hierarchy (DIT is large) also leads to greater design complexity.
- On the positive side, **large DIT values imply that many methods may be reused.**

Number of children (NOC):-

- The subclasses that are immediately subordinate to a class in the class hierarchy are termed its *children*.
- Referring to previous figure, class **C2** has three children—subclasses **C21**, **C22**, and **C23**.
- As the number of children grows, reuse increases, the abstraction represented by the parent class can be diluted.
- In this case, some of the children may not really be appropriate members of the parent class.
- **As NOC increases, the amount of testing (required to exercise each child in its operational context) will also increase.**

4. Coupling between object classes (CBO):

- The CRC model may be used to determine the value for CBO
- CBO is the number of collaborations listed for a class on its CRC index card.
- **As CBO increases, it is likely that the reusability of a class will decrease.**
- **If values of CBO is high, then modification get complicated.**
- **Therefore, CBO values for each class should be kept as low as is reasonable.**

5. Response for a class (RFC)

- Response for a class is “a set of methods that can potentially be executed in response to a message received by an object of that class”
- RFC is the number of methods in the response set.
- **As RFC increases, the effort required for testing also increases because the test sequence grows.**
- As RFC increases, the overall design complexity of the class increases.

6. Lack of cohesion in methods (LCOM).

- LCOM is the number of methods that access one or more of the same attributes.
- If no methods access the same attributes, then $LCOM = 0$.
- To illustrate the case where $LCOM \neq 0$, consider a class with six methods.
- Four of the methods have one or more attributes in common (i.e., they access common attributes). Therefore, $LCOM = 4$.
- **If LCOM is high, methods may be coupled to one another via attributes. This increases the complexity of the class design.**
- In general, high values for LCOM imply that the class might be better designed by breaking it into two or more separate classes.
- It is desirable to keep cohesion high; that is, keep LCOM low.

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).
- Not depend on programming.
- Ex: effort expanded/use case

Metrics for software quality

- Measuring Quality
 - It consist of 4 parameter.
 - Correctness
 - Maintainability
 - Integrity
 - Usability
- Defect Removal Efficiency method

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.
- The most common measure for correctness is **defects per KLOC**, where a defect is defined as a verified lack of conformance to requirements.
- When considering the overall quality of a software product, defects are those problems reported by a user of the program

Maintainability

- Maintenance required more effort than any other software engineering activity.
- 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 requirement.
- There is no way to measure maintainability directly; therefore, we must use indirect measures.
- A simple **time-oriented metric 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*.

- Another method is, **cost-oriented metric** for maintainability called **spoilage** - the cost to correct defects encountered after the software has been released to its end-users.
- By determining spoilage ratio to overall cost is plotted as a function time.
- **Project manager can determine, overall maintainability of software produced by a software development team**

Integrity

- Software integrity has become increasingly important in the age of hackers and firewalls.
- This attribute measures a system's ability to withstand attacks (both accidental and intentional) to its security.
- Attacks can be made on all three components of software:
 - Programs
 - Data
 - Documents
- To measure integrity, two additional attributes must be defined:
 - Threat
 - Security

- **Threat** is the probability (which can be estimated or derived from practical evidence) that an attack of a specific type will occur within a *given time*.
- **Security** is the probability (which can be estimated or derived from practical evidence) that the attack of a specific type will be prevented.
- Integrity of a system can then be defined as

$$\text{integrity} = \text{summation} [(1 - \text{threat}) \times (1 - \text{security})]$$

where threat and security are summed over each type of attack.

Usability

- The phrase "user-friendliness" has become everywhere in discussions of software products.
- If a program is not user-friendly, it is often doomed to failure, even if the functions that it performs are valuable.
- Usability is an attempt to quantify user-friendliness and can be measured in terms of four characteristics:
 - the physical and or intellectual skill required to learn the system,
 - the time required to become moderately efficient in the use of the system
 - productivity measured when the system is used by someone who is moderately efficient
 - A subjective assessment (sometimes through a questionnaire) of users attitudes toward the system.

Defect Removal Efficiency

- A quality metric that provides benefit at both the project and process level is defect removal efficiency (DRE).
- DRE is a measure of the filtering ability of quality assurance and control activities as they are applied throughout all process framework activities.
- To compute DRE:
 - $DRE = E / (E + D)$

Where E = no. of error before release and D = defect found after release of software to end users

Defect Removal Efficiency

- The ideal value for DRE is 1. That is, no defects are found in the software.
- Realistically, D will be greater than 0, but the value of DRE can still approach 1. As E increases (for a given value of D), the overall value of DRE begins to approach 1.
- In fact, as E increases, it is likely that the final value of D will decrease (errors are filtered out before they become defects).
- DRE encourages a software project team to institute techniques for finding as many errors as possible before delivery.

- DRE can also be used within the project to assess a team's ability to find errors before they are passed to the next framework activity or software engineering task.

For example, the requirements analysis task produces an analysis model that can be reviewed to find and correct errors. Those errors that are not found during the review of the analysis model are passed on to the design task.

- When used in this context, we redefine DRE as

$$DRE_i = E_i / (E_i + E_{i+1})$$

E_i is the number of errors found during software engineering activity i . E_{i+1} number of errors found during software engineering activity $i+1$

- A quality objective for a software team is to achieve DRE_i that approaches 1. That is, errors should be filtered out before they are passed on to the next activity.

Web app project metrics

- No of static web pages
- No of dynamic web pages
- No of internal page links
- No of persistent data objects
- No of external system interfaced
- No of static content objects(graphics)
- No of dynamic content objects
- No of executable functions
- Customization index(c)= $N_{dp}/N_{dp}+N_{sp}$ ($0 < c \leq 1$)

Empirical Estimation Models

- Source Lines of Code (SLOC)
- Function Point (FP)
- Constructive Cost Model (COCOMO)

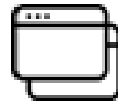
SLOC

- The **project size** helps to determine the resources, effort, and duration of the project.
- **SLOC is defined as the Source Lines of Code that are delivered as part of the product**
- The **effort spent on creating the SLOC is expressed in relation to thousand lines of code (KLOC)**
- **This technique includes the calculation of Lines of Code, Documentation of Pages, Inputs, Outputs, and Components of a software program**
- The SLOC technique is **language-dependent**
- The **effort required to calculate SLOC may not be the same for all languages**

Software Development Project Classification

Based on the development complexity

Organic



Application programs

e.g. data processing programs

A **development project** can be considered of **organic** type, if the project deals with **developing** a **well understood application program**, the **size** of the **development team** is reasonably **small**, and the **team members** are **experienced** in **developing similar types of projects**

Semidetached



Utility programs

e.g. Compilers, linkers

A **development project** can be considered of **semidetached** type, if the **development consists** of a **mixture** of **experienced & inexperienced staff**. Team members may have **limited experience on related systems** but may be unfamiliar with some aspects of the system being developed.






Embedded

System programs

e.g. Operating systems, real-time systems

A **development project** is considered to be of **embedded** type, if the **software** being developed is **strongly coupled** to **complex hardware**, or if the **strict regulations** on the **operational procedures** exist

Model	Project Size	Nature of Project	Innovation	Dead Line	Development Environment
Organic 	Typically 2-50 KLOC	Small Size Project, Experienced developers in the familiar environment, E.g. Payroll, Inventory projects etc.	Little	Not Tight	Familiar & In-house
Semi Detached 	Typically 50-300 KLOC	Medium Size Project, Medium Size Team, Average Previous Experience, e.g. 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. E.g. ATMs, Air Traffic Controls	Significant Required	Tight	Complex hardware & customer Interfaces

COCOMO Model

- **COCOMO (Constructive Cost Estimation Model)** was proposed by Boehm
- According to Boehm, **software cost estimation** should be done through three stages:
- Basic COCOMO,
- Intermediate COCOMO,
- Complete COCOMO

Basic COCOMO Model

- The **basic COCOMO** model gives an **approximate estimate** of the project parameters.
- The **basic COCOMO estimation** model is given by the following expressions

$$\textit{Effort} = a_1 + (KLOC)^{a_2} PM \quad \textit{Tdev} = b_1 \times (\textit{Effort})^{b_2} \textit{Months}$$

Basic cocomo model

- **KLOC** is the estimated size of the software product expressed in thousands Lines of Code,
- **a** , **a** , **b** , **b** are constants for each category of software products,
- **Tdev** is the estimated **time to develop** the software, **expressed in months**,
- **Effort** is the total effort required to develop the software product, expressed in **person months (PMs)**.

Basic cocomo model

- The effort estimation is expressed in **units of person-months (PM)**
- It is the **area under the person-month plot** (as shown in fig.)
- An **effort of 100 PM does not** imply that **100 persons** should **work for 1 month**
- **does not** imply that **1 person** should be **employed for 100 months**
- **it denotes the area under the person-month curve** (fig.)

Basic cocomo model

- Every line of source text should be calculated as one LOC irrespective of the actual number of instructions on that line
- If a single instruction spans several lines (say n lines), it is considered to be n LOC
- The values of a , a , b , b for different categories of products (i.e. organic, semidetached, and embedded) as given by Boehm
- He derived the expressions by examining historical data collected from a large number of actual projects

Example of basic cocomo model

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

$\text{Effort} = a_1 + (KLOC)^{a_2} \text{ PM}$	$\text{Tdev} = b_1 \times (\text{Effort})^{b_2} \text{ Months}$
$= 2.4 + (32)^{1.05} \text{ PM}$	$= 2.5 \times (91)^{0.38} \text{ Months}$
$= 91 \text{ PM}$	$= 14 \text{ Months}$

Continue..

Cost required to develop the product = $14 \times 15000 = \text{Rs. } 2,10,000/-$

INTERMEDIATE MODEL

- Extension of basic model
- Set of 15 additional predictors(Cost Drivers)
- Cost driver adjust nominal cost of project to actual project.
- 1) Product attributes(3): data base size, reliability, complexity
- 2) Computer attributes(4): execution time ,main storage, virtual memory, turn around time
- 3) Personnel attributes(5):analysis capability, language experience, programmer capability, application experience
- 4) Project attributes(3):modern programming ,use of tools, requirement development schedule

- $E = a_i (KLOC)^{b_i} * EAF(\text{effort adjustment factor})$
- $D = c_i (E_i)^{d_i}$ (development time)

Project	a_i	b_i	c_i	d_i
organic	3.2	1.05	2.5	0.38
semidetached	3.0	1.12	2.5	0.35
embedded	2.8	1.20	2.5	0.32

Completer or detailed COCOMO model

- Large team
- Complex project
- Experience and creative members are required for divide project in to modules and apply cocomo to all .

Phases of cocomo:

- 1)planning & requirement
- 2)system design
- 3)detailed design
- 4)Module code and test
- 5) Intigration and test
- 6)cost cunstruction model

- **Note :**
- Refer page number 709 to 712 of pressman 7th edition for example of COCOMO II model
- $$NOP(\text{number of object points}) = \text{object points} * [(100 - \%reuse) / 100]$$
- $$Prod = NOP / \text{person-month}$$
- $$\text{Estimated effort} = NOP / PROD$$

Risk

- **risk** is a **potential (probable) problem** – which might **happen and might not**
- **Conceptual definition of risk**
- Risk concerns future happenings
- Risk involves change in mind, opinion, actions, places, etc.
- Risk involves choice and the uncertainty that choice entails
- **Two characteristics of risk**
- **Uncertainty:**
- The risk **may or may not happen**, so there are no 100% risks (some of those may called constraints)
- **Loss**
- If the **risk** becomes a **reality** and **unwanted consequences** or **losses** occur

Risk Categorization: Approach-1

- **Project risks**
- They **threaten** the **project plan**
- If they become real, it is likely that the **project schedule will slip** and that **costs will increase**
- **Technical risks**
- They **threaten the quality and timeliness** of the software to be produced
- If they become real, **implementation may become difficult** or impossible
- **Business risks**
- They **threaten the feasibility** of the **software** to be built
- If they become real, they **threaten the project** or the **product**

Sub-categories of Business risks

- **Market risk**
Building an **excellent product** or system that **no one really wants**
- **Strategic risk**
Building a **product that no longer fits into** the overall **business strategy** for the company
- **Sales risk**
Building a **product** that the **sales force doesn't understand** how to sell
- **Management risk**
Losing the support of senior management due to a change in focus or a change in people
- **Budget risk**
Losing budgetary or personnel commitment

Risk categorization approach -2

- **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)
- **Predictable risks**
- Those **risks** that are **deduced** (draw conclusion) from **past project** experience (Ex. past turnover)
- **Unpredictable risks**
- Those **risks** that can and do **occur**, but are **extremely difficult** to **identify in advance**

Risk Strategies (Reactive vs. Proactive)

- **Reactive risk strategies**
- **“Don't worry, I will think of something”.**
- The **majority of software teams and managers rely** on this approach
- **Nothing is done about risks until something goes wrong**
- The **team then flies into action** in an attempt to **correct the problem rapidly** (fire fighting)
- **Crisis management** is the **choice of management techniques**
- **Proactive risk strategies**
- **Steps for risk management are followed**
- Primary **objective** is to **avoid risk** and to have an **emergency plan in place** to **handle** unavoidable **risks** in a **controlled and effective manner**

Steps for Risk Management

- 1. **Identify** possible **risks** and recognize what can go wrong
- 2. **Analyze** each **risk** to estimate the probability that it will occur and the impact (i.e., damage) that it will do if it does occur
- 3. **Rank** the **risks** by probability and impact. Impact may be negligible, marginal, critical, and catastrophic.
- 4. **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**.
- **Plan includes identify** known and predictable **risks**, the project manager **takes a first step** toward, avoiding them when possible controlling them when necessary
- **Generic Risks**
 - **Risks** that are a potential **threat** to **every** software **project**
 - **Product-specific Risks**
 - Risks that can be **identified only by clear understanding** of the technology, the people and the environment, that is **specific to the software** that is to be built

Known and Predictable Risk Categories

- One method for identifying risks is **to create a risk item checklist**
- The checklist can be used for risk identification which **focuses on some subset of known and predictable risks** in the following generic subcategories:
- **Product Size:** risks associated with overall size of the software to be built
- **Business Impact:** risks associated with constraints imposed by management or the marketplace
- **Customer Characteristics:** risks associated with sophistication of the customer and the developer's ability to communicate with the customer in a timely manner

Known and Predictable Risk Categories

- **Process Definition:** risks associated with the degree to which the software process has been defined and is followed
- **Development Environment:** risks associated with availability and quality of the tools to be used to build the project
- **Technology to be Built:** risks associated with system to be built and the “newness” of the system complexity of the technology.
- **Staff Size and Experience:** risks associated with overall technical and project experience of the software engineers who will do the work

Risk Estimation (Projection)

- **Risk projection (or estimation) attempts to rate each risk in two ways**
- The **probability** that the **risk is real**
- The consequence (**effect**) of the **problems** associated with the risk
- **Risk Projection/Estimation Steps**
- **Establish a scale that reflects the perceived likelihood (probability) of a risk.**
- *Ex., 1-low, 10-high*
- Explain the **consequences of the risk**
- **Estimate the impact of the risk** on the project and product.
- **Note the overall accuracy of the risk projection** so that there will be **no misunderstandings**

RMMM

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

RMMM

- **Risk Mitigation** is a problem avoidance activity
- **Risk Monitoring** is a project tracking activity
- **Risk Management** includes contingency plans that risk will occur

RMMM con....

- Risk **mitigation (avoidance)** is the **primary strategy** and is achieved through a plan
- **For Ex., Risk of high staff turnover**
- To mitigate this risk, you would develop a strategy for reducing turnover. The possible steps to be taken are:
- **Meet with current staff** to determine **causes for turnover** (e.g., poor working conditions, low pay, and competitive job market)
- **Mitigate** those **causes** that are **under your 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**

RMMM con...

- **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**

RMMM PLAN

- The RMMM **PLAN documents** all **analysis** and used by the project **project plan**
- work performed as part of **risk** manager as part of the **overall**
- 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.
- The format of RIS is describe in diagram

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	

Software 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.
- In addition, estimates should attempt to define best-case and worst-case scenarios so that project outcomes can be bounded. Although there is an inherent degree of uncertainty, the software team embarks on a plan that has been established as a consequence of these tasks.
- Therefore, the plan must be adapted and updated as the project proceeds.

- *Task Set for Project Planning*

- Establish project scope.
- Determine feasibility.
- Analyze risks.
- Define required resources.
 - Determine required human resources.
 - Define reusable software resources.
 - Identify environmental resources.
- Estimate cost and effort.
 - Decompose the problem.
 - Develop two or more estimates using size, function points, process tasks, or use cases.
 - Reconcile the estimates.
- Develop a project schedule.
 - Establish a meaningful task set.
 - Define a task network.
 - Use scheduling tools to develop a time-line chart.
 - Define schedule tracking mechanisms.

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.
- It is important to note, however, that the schedule evolves over time.
- During early stages of project planning, a macroscopic schedule is developed.
- This type of schedule identifies all major process framework activities and the product functions to which they are applied.

Scheduling Principles

- Compartmentalization
- The product and process must be decomposed into a manageable number of activities and tasks
- Interdependency
- Tasks that can be completed in parallel must be separated from those that must be completed serially
- Time allocation
- Every task has start and completion dates that take the task interdependencies into account

- Effort validation
- Project manager must ensure that on any given day there are enough staff members assigned to completed the tasks within the time estimated in the project plan
- Defined Responsibilities
- Every scheduled task needs to be assigned to a specific team member
- Defined outcomes
- Every task in the schedule needs to have a defined outcome (usually a work product or deliverable)
- Defined milestones
- A milestone is accomplished when one or more work products from an engineering task have passed quality review

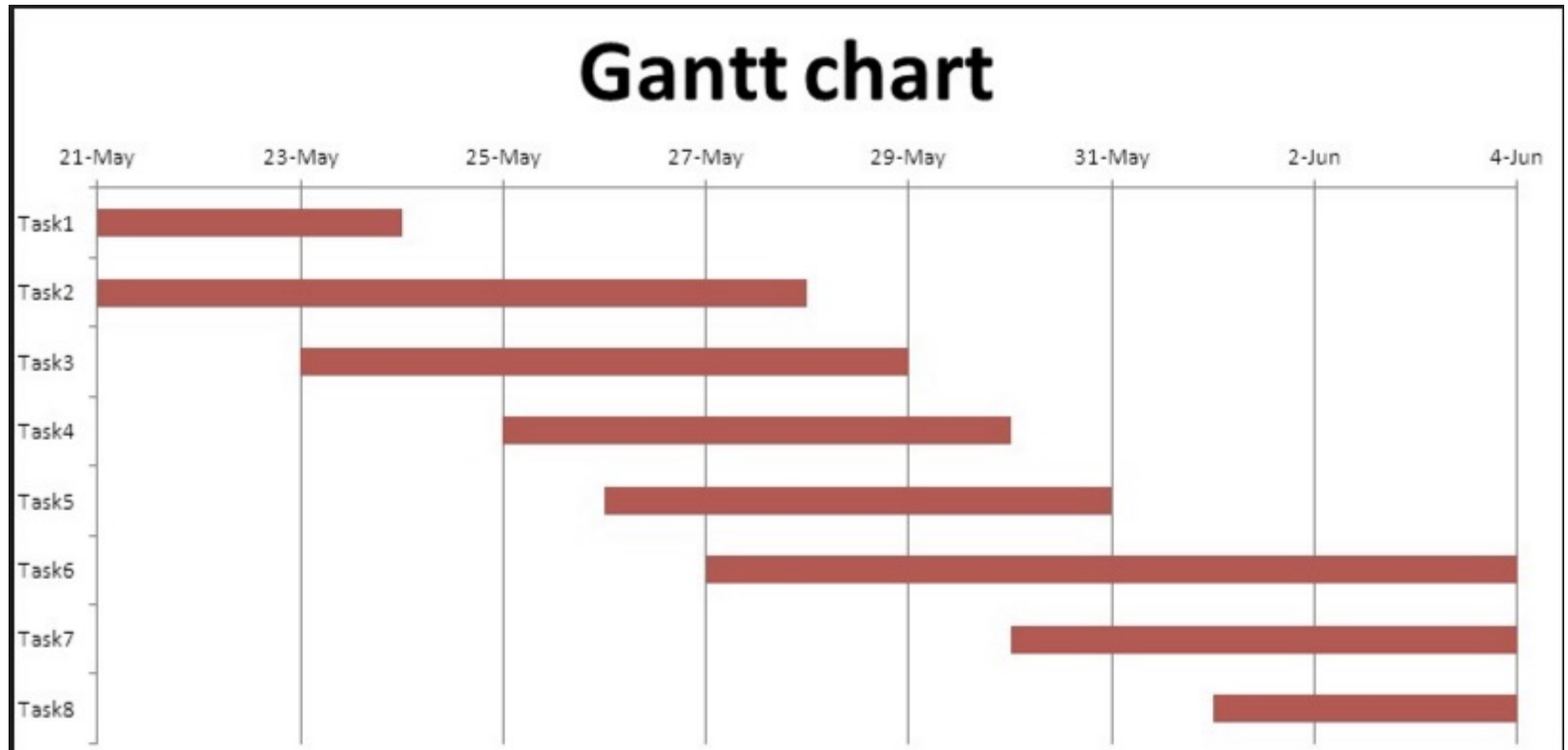
Scheduling

- 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 for software projects.
- Program evaluation and review technique (PERT) and the critical path method (CPM) are two project scheduling methods that can be applied to software development.

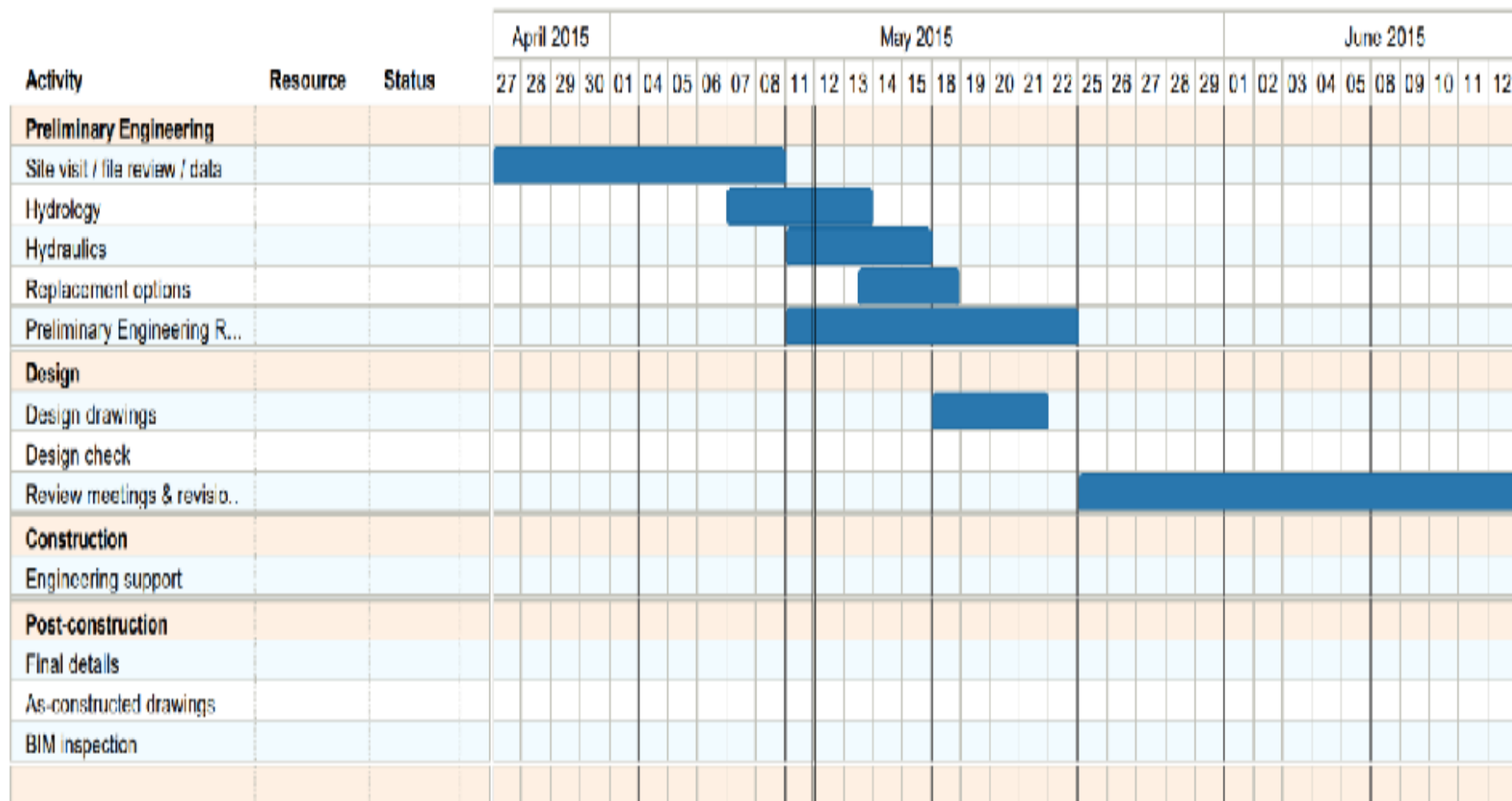
Scheduling

- 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.
- 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.

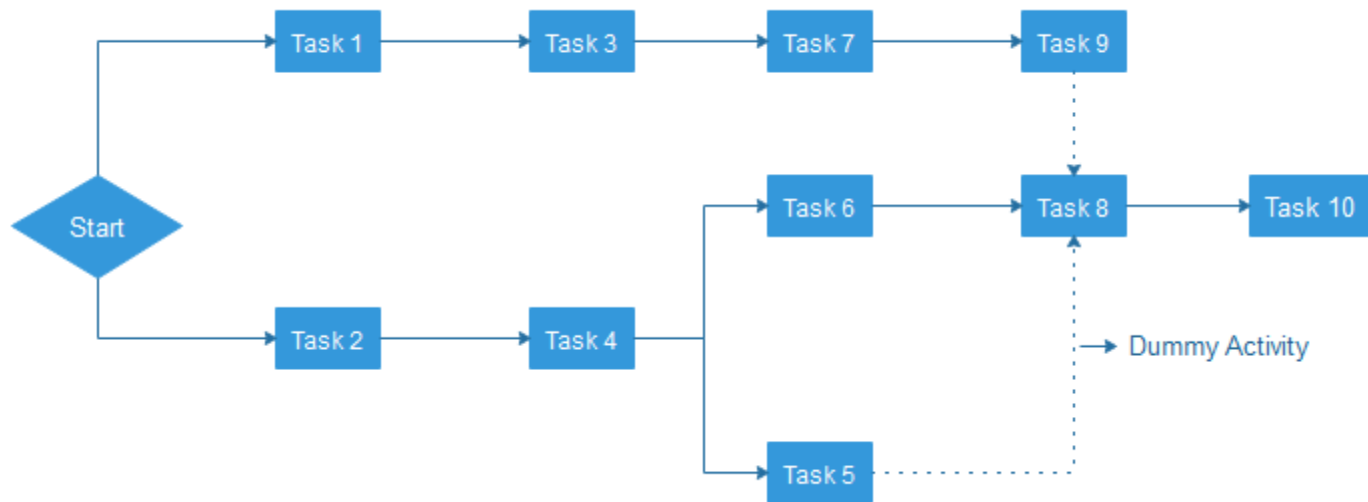
Sample Gantt chart-1



Gantt Chart-2



PERT chart



Assignment

- What do you mean by risk? What is software risk? Explain all type of Software risk.
- Write short note on : Risk Management or RMMM
- Explain Software Project Management and W⁵HH principles.
- What is software measurement? Explain Software matrices used for software cost estimation. Or Explain Software matrices in details.
- Explain software project planning.
- Explain project scheduling process and Gantt chart in detail.

References

- <https://www.forecast.app/blog/benefits-of-using-project-management-software>
- <https://www.youtube.com/watch?v=z9KjqfpR9XI>
- JIRA <https://youtu.be/aP7W7zNTM2I>
- Risk management
<https://www.youtube.com/watch?v=OlzMrXtgl1I>