SOFTWARE ENGINEERING with 203124253

Prof. Jignasha Parmar, Assistant Professor
Information and Technology

UNIT-2

Software Project Management

Planning a Software Project

Software Project Management

•Management Spectrum, People – Product – Process- Project, W5HH Principle, Importance of Team Management

Planning a Software Project

Scope and Feasibility, Effort Estimation, Schedule and staffing, Quality Planning, Risk management- identification, assessment, control, project monitoring plan, Detailed Scheduling

W^5HH of Project Management

Boehm suggests an approach (W5HH) that addresses project objectives, milestones and schedules, responsibilities, management and technical approaches and required resources

Why is the system being developed?

Enables all parties to assess the validity of business reasons for the software work. In another words - does the business purpose justify the expenditure of people, time, and money?

What will be done?

The answers to these questions help the team to establish a project schedule by identifying key project tasks and the milestones that are required by the customer

When will it be accomplished?

Project schedule to achieve milestone

W^5HH of Project Management Cont.

Who is responsible?
Role and responsibility of each member

Where are they organizationally located?
Customer, end user and other stakeholders also have responsibility

How will the job be done technically and managerially?
Management and technical strategy must be defined

How much of each resource is needed?
Develop estimation

W^5HH

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

Terminologies

Measure

It provides a **quantitative indication** of the extent (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 (limit) to which a system, component or process possesses (obtain) a given attribute

It **relates individual measures** in some way

Ex., number of errors found per review

Direct Metrics

Immediately measurable attributes

Ex., Line of Code (LOC), Execution Speed, Defects Reported

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

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.

Metric Classification Base

- **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.
- **Project**
  - Software development work in which a software process is used
  - The actual act of executing the activities for some specific user needs
- **Product**
  - **The outcomes of** a software **project**
  - All the outputs that are produced while the activities are being executed

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** are **collected across all projects** and over **long periods of time**
- 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 Cont.

- 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**

Project Metrics

- Project **metrics enable** a software project **manager** to,
  - **Assess the status** of an ongoing project
  - **Track** potential **risks**
  - **Uncover problem areas** before their status becomes critical
  - **Adjust work flow** or tasks
  - **Evaluate** the project **team's ability** to **control quality** of software work products
- Many of the same metrics are used in both the process and project domain
- **Project metrics** are **used** for making **tactical (smart) decisions**
- They are **used** to adapt **project workflow** and **technical activities**

Project Metrics Cont.

•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**
  - Ex., Code Complexity Metric

Types of Measures

Categories of Software Measurement

Direct measures of the
Software process Ex., cost, effort, etc.
Software product Ex., lines of code produced, execution speed, defects reported, etc.

Indirect measures of the
Software product Ex. functionality, quality, complexity, efficiency, reliability, etc.

Software Measurement

Metrics for Software **Cost** and **Effort estimations**
Size Oriented Metrics
Function Oriented Metrics
Object Oriented Metrics
Use Case Oriented Metrics

Size-Oriented Metrics

- **Derived** by **normalizing** (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 Cont.

- 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**
  - **Penalize well-designed** but **short programs**
  - **Cannot** easily **accommodate nonprocedural** languages
  - **Require** a **level of detail** that may be **difficult to achieve**

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** = Count Total * [0.65 + 0.01 * Sum (Value Adjustment Factors)]
- Function Point **values on past projects** can be **used** to compute,
  - for example, the **average number of lines of code** per function point

Function Oriented Metrics Cont.

- **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**

Function Point Metrics

- The **function point (FP)** metric can be **used** effectively as a **means for measuring the functionality** delivered by a system
- Using **historical data**, the **FP metric** can be **used** to
  - **Estimate the cost** or **effort** required to design, code, and test the software
  - **Predict the number of errors** that will be encountered during testing
  - **Forecast** the **number of components** and/or the **number of projected source lines** in the implemented system

Function Point Components Cont..

Information domain values (components) are defined in the following manner

- **Number of external inputs (EIs)**
  input data originates from a user or is transmitted from another application
- **Number of external outputs (EOs)**
  external output is derived data within the application that provides information to the user output refers to reports, screens, error messages, etc.
- **Number of external inquiries (EQs)**
  external inquiry is defined as an online input that results in the generation of some immediate software response in the form of an online output
- **Number of internal logical files (ILFs)**
  internal logical file is a logical grouping of data that resides within the application's boundary and is maintained via external inputs
- **Number of external interface files (EIFs)**
  external interface file is a logical grouping of data that resides external to the application but provides information that may be of use to the another application

Compute Function Points

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

Count Total is the sum of all FP entries

Fi (i=1 to 14) are complexity value adjustment factors (**VAF**).

Value adjustment factors are used to provide an indication of problem complexity

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

Compute Function Points

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

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

Study of requirement specification for a project has produced following results
Need for **7 inputs**, **10 outputs**, **6 inquiries**, **17 files** and **4 external interfaces**
Input and **external interface function point** attributes are of **average complexity** and all **other function points** attributes are of **low complexity**
Determine **adjusted function points** assuming complexity **adjustment value is 32**.

Compute Function Points Cont.

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

Compute Function Points Cont.

Function Point Calculation Example

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

Compute Function Points Cont.

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 = 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

Compute Function Points Cont.

Function Point Calculation Example 2

Need for **7 inputs**, **10 outputs**, **6 inquiries**, **17 files** and **4 external interfaces**

Input and **external interface function point** attributes are of **average complexity** and all **other function points** attributes are of **low complexity**

Study of requirement specification fo

Input and **external interface fu**

complexity and all **other function**

Determine **adjusted function points** assuming complexity **adjustment value is 32**.

Determine **adjusted function po**

val

Software Project Estimation

It can be **transformed** from a **black art** to a **series of systematic steps** that provide **estimates** with **acceptable risk**

To **achieve** reliable **cost** and **effort estimates**, a number of options arise:

- Delay estimation **until late in the project** (obviously, we can achieve 100 percent accurate estimates after the project is complete!)
- Base **estimates** on **similar projects** that have already been completed
- Use relatively simple **decomposition techniques** to generate project cost and effort estimates
- **Use** one or more **empirical models** for software cost and effort estimation.

Software Project Estimation

- Software **project estimation** is a form of **problem solving** and in most cases, the problem to be solved is **too complex to be considered in one piece**
- For this reason, **decomposing the problem**, re-characterizing it as a set of smaller problems is required
- Before an estimate can be made, the **project planner** must **understand the scope of the software** to be built and must generate an estimate of its "size"

Decomposition Techniques

1. Software Sizing
2. Problem based Estimation
   LOC (Lines of Code) based,
   FP (Function Point) based
3. Process based Estimation
4. Estimation with Use-cases

Software Sizing

Putnam and **Myers** suggest **four** different **approaches** to the **sizing problem**

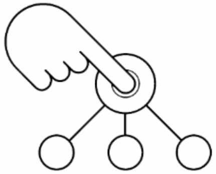
"Fuzzy logic" sizing

- This approach uses **the approximate reasoning techniques** that are the cornerstone of fuzzy logic.

Function Point sizing

- The planner develops **estimates of the information domain characteristics**

Standard Component sizing



- Estimate the **number of occurrences** of each **standard component**
- Use **historical** project **data** to determine the **delivered LOC** size per standard component.

Software Sizing Cont..

Change sizing

- **Used** when **changes** are being **made** to **existing software**
- Estimate the **number** and **type of modifications** that must be accomplished
- An **effort ratio** is then **used** to **estimate** each **type of change** and the **size of** the **change**

Problem Based Estimation

- **Start** with a **bounded** statement of **scope**
- **Decompose** the **software** into **problem functions** that can each be **estimated individually**
- **Compute** an **LOC** or **FP** value for **each function**
- **Derive cost** or **effort estimates** by applying the **LOC** or **FP** values to your **baseline productivity metrics**
  - Ex., LOC/person-month or FP/person-month
- **Combine function estimates** to produce an **overall estimate** for the **entire project**
- In general, the **LOC/pm** and **FP/pm** metrics should be computed by project domain
  - **Important factors** are **team size**, **application area** and **complexity**

Problem Based Estimation

- **LOC** and **FP** estimation **differ in the level** of **detail** required for **decomposition** with each value
  - For **LOC**, **decomposition of functions** is **essential** and should go into considerable detail **(the more detail, the more accurate the estimate)**
  - For **FP**, **decomposition** occurs for the **five** information **domain characteristics** and the **14 adjustment factors**
    - **External Inputs, External Outputs, External Inquiries, Internal Logical Files, External Interface Files**
- For **both approaches**, the planner **uses lessons learned** to **estimate**,
  - An **optimistic (S_{opt})**, **most likely (S_m)**, and **pessimistic (S_{pess})** estimates Size (S) value for each function or count
  - Then the expected Size value S is computed as
    - $S = (S_{opt} + 4\ S_m + S_{pess})/6$

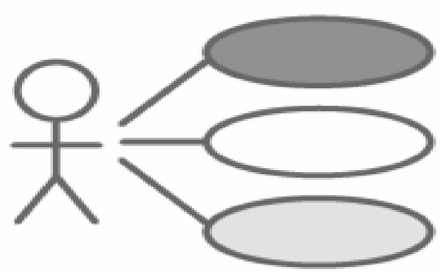
Historical LOC or FP data is then compared to S in order to cross-check it.

Process Based Estimation

- **Identify** the **set of functions** that the software needs to perform as obtained **from the project scope**
- **Identify** the **series of framework activities** that need to be performed **for each function**
- **Estimate the effort** (in **person months**) that will be **required to accomplish** each software process **activity** for **each function**
- **Apply average labor** rates (i.e., **cost/unit** effort) to the **effort estimated** for each process activity
- **Compute** the **total cost** and **effort** for each function and each framework activity.
- **Compare the resulting values** to those obtained by way of the **LOC and FP** estimates
- If **both** sets of **estimates agree**, then your numbers are **highly reliable**
- **Otherwise**, conduct **further investigation** and **analysis** concerning the **function** and **activity breakdown**

Process Based Estimation Cont..

- **Before use cases** can be used for **estimation**,
  - the **level within the structural hierarchy** is established,
  - the **average length (in pages) of each use case** is determined,
  - the **type of software** (e.g., real-time, business, engineering/scientific, WebApp, embedded) is defined, and
  - a **rough architecture** for the system is **considered**
- Once these characteristics are established,
  - empirical data may be used to establish the estimated number of LOC or FP per use case (for each level of the hierarchy).
- Historical data are then used to compute the effort required to develop the system.



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

Software Development Project Classification

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

Software Development Project Cont.

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, and
- Complete COCOMO

Basic COCOMO Model

$$Effort = a_1 * (KLOC)^{a_2}\ PM$$

$$Tdev = b_1 \times (Effort)^{b_2}\ Months$$

The **basic COCOMO** model gives an **approximate estimate** of the project parameters

- **KLOC** is the estimated size of the software product expressed in Kilo Lines of Code,
- $\mathbf{a_1, a_2, b_1, b_2}$ 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)**.

Project	A1	A2	B1	B2
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Basic COCOMO Model Cont.

- The effort estimation is expressed in **units of person-months (PM)**
- It is the **area under the person-month plot**
- 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**

- **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 **nLOC**
- The values of $\mathbf{a_1}$, $\mathbf{a_2}$, $\mathbf{b_1}$, $\mathbf{b_2}$ 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

Basic COCOMO Model Cont.

$$\boldsymbol{Effort} = a_1 * (KLOC)^{a_2}\ PM$$
$$= 2.4 * (32)^{1.05}\ PM$$
$$= 91\ PM$$

$$\boldsymbol{Tdev} = b_1 \times (Effort)^{b_2}\ Months$$
$$= 2.5 \times (91)^{0.38}\ Months$$
$$= 14\ Months$$

Example:

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**

Cost required to develop the product = 14 x 15000 = Rs. **2,10,000/-**

Intermediate COCOMO Model Cont.

- The **basic COCOMO** model **assumes** that **effort** and **development** time **are functions** of the **product size alone**
- However, a **host of other** project **parameters** besides the product size **affect** the **effort required** to develop the product as well as the **development time**
- Therefore, **in order to obtain an accurate estimation** of the effort and project **duration**, the **effect** of all relevant **parameters** must be **taken** into **account**
- The **intermediate COCOMO** model **recognizes this fact** and refines the initial estimate obtained using the basic COCOMO expressions **by using a set of 15 cost drivers (multipliers)** based on **various attributes** of software development

The cost drivers can be **classified** as being attributes **of the following items**

1.Computer **2.Product** **3.Personnel** **4.Development Environment**

Complete COCOMO Model Cont.

- 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**
- The **complete COCOMO** model **considers** these differences in **characteristics** of the **subsystems** 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**

Project Scheduling & Tracking

Scheduling Principles

- Compartmentalization
- Interdependency
- Time Allocation
- Effort Validation
- Define Responsibilities
- Define Outcomes
- Define Milestones

Effort Distribution

General guideline: **40-20-40 rule**

- **40%** or more of all effort allocated to **analysis and design tasks**
- **20%** of effort allocated to **programming**
- **40%** of effort allocated to **testing**

Although most software organizations encounter the following **projects types**:

1. **Concept Development**
2. **New Application Development**
3. **Application Enhancement**
4. **Application Maintenance**
5. **Reengineering**

Scheduling methods

1. **Program Evaluation and Review Technique** (**PERT**)
2. **Critical Path Method** (**CPM**)

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

Project Schedule Tracking

- Several ways to track a project schedule:
  - Conducting **periodic project** status **meeting**
  - **Evaluating** the **review results** in the software process
  - **Determine** if formal **project milestones** have been **accomplished**
  - **Compare actual start date** to **planned start date** for each task
  - Informal **meeting with practitioners**
  - Using **earned value analysis** to **assess progress** quantitatively
- **Project manager** takes the **control** of the **schedule** in the aspects of
  - Project Staffing, Project Problems, Project Resources, Reviews, Project Budget

Risk analysis & Management

A **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 ⟵⟶ **Loss**

The risk **may** or **may not happen**, so there are no 100% risks (some of those may called constraints)

If the **risk** becomes a **reality** and **unwanted consequences** or **losses** occur

Risk Categorization: Approach-1

- Project risks
- Technical risks
- Business risks

Sub-categories of Business risks

- Market risk
- Strategic risk
- Sales risk
- Management risk
- Budget risk

Risk Categorization: Approach-2

- **Known risks**
  - Those **risks** that can be **uncovered after careful evaluation** of
- **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**

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 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**

Risk Identification

- Risk identification is a **systematic attempt** to **specify threats** to the project **plan**
- By **identifying** 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

- Product Size
- Business Impact
- Customer Characteristics
- Process Definition
- Development Environment
- Technology to be Built
- Staff Size and Experience

Parul® University

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

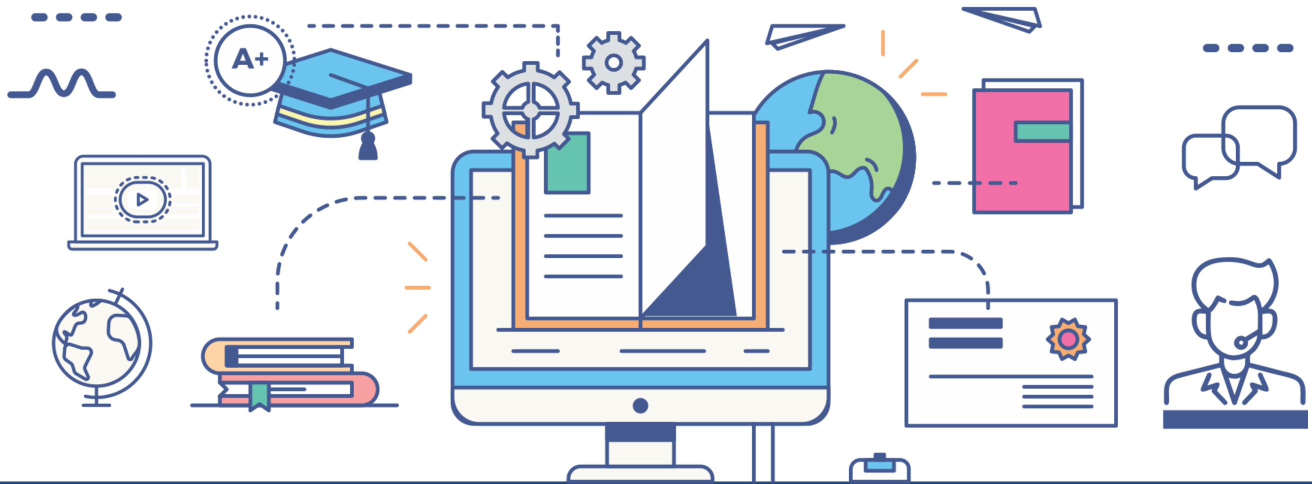
- **Establish a scale** that reflects the **perceived** likelihood **(probability**) of a **risk**. *Ex., 1-low, 10-high*
- Explain the **consequences of the risk**
- **Estimate** the **impact of the risk** on the project and product.
- **Note** the **overall accuracy of the risk projection** so that there will be **no misunderstandings**

Parul® University

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**

DIGITAL LEARNING CONTENT



Parul® University