Software Engineering

→ Measures, Matrices and ... + Fig 4.1 + description

4.1 Measures, Metrics and Indicators

① Measure provides a quantitative indication of the extent, amount, dimension, capacity or size of some attribute of a product or process. Measurement is the act of determining a measure.

② Metric as a quantitative measure of the degree to which a system, component, or process processes a given attribute.

③ A software engineer collects measures and develops metrics so that indicators will be obtained. An indicator is a metric or combination of metrics that provide insight into the software process, project or product itself that enables the project manager or software engineers to adjust the process, project to make things better

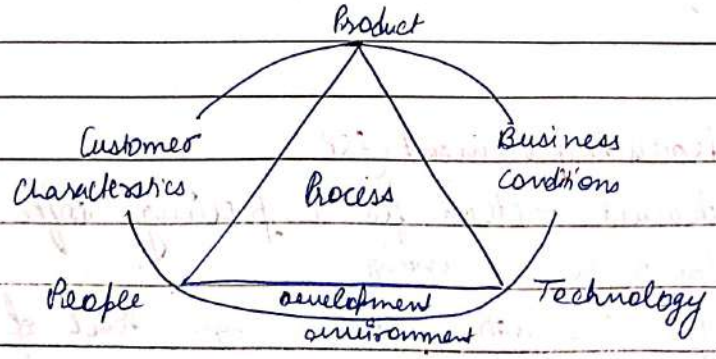
4.2 Metrics in the Process and Project Domains.

Process Metrics and Software Improvement

The only rational way to improve any process is to measure specific attributes of the process, develop a set of meaningful metrics based on these attributes, then use the metrics to provide indicators that will lead to a strategy for improvement.

Determinants for software quality and organizational effectiveness.

Explains



(1) Private vs Public Process Metrics :

(1) Private Metrics

(a) Used by individuals to monitor and improve their personal performance

(b) Examples : defect rates by individual, errors found during develop.

(c) These should not be used to evaluate or appraise individual engineers

(2) Public Metrics

(a) Used by teams, but only visible within the team

(b) Examples → defects reported for functions developed by a team, errors found during technical reviews.

(c) Help teams identify areas for improvement.

(2) Personal software Process (PSP) :

(a) Developed by Humphrey

(b) A structured approach to help engineers improve their personal work performance.

(c) Involves :

(a) forms, scripts and standards to help plan and estimate work

(b) Methods to define processes and measure quality / productivity

~~(d) Prompt~~

(3) Software Metrics Etiquette

(a) use metrics with common sense and organizational sensitivity

(b) Provide feedback to individuals & team

(c) .

(4) Statistical Software Process Improvement (SSPI)

(a) A more advanced method for improving software process

(b) Involves software failure-analysis

(c) Suitable for organizations with a higher level of process maturity

Failure analysis

SSPI uses software failure analysis to collect info about all errors. Failure analysis works in the following manner:

(a) All errors and defects are categorized by origin.

(b) The cost to correct each error and defect is recorded

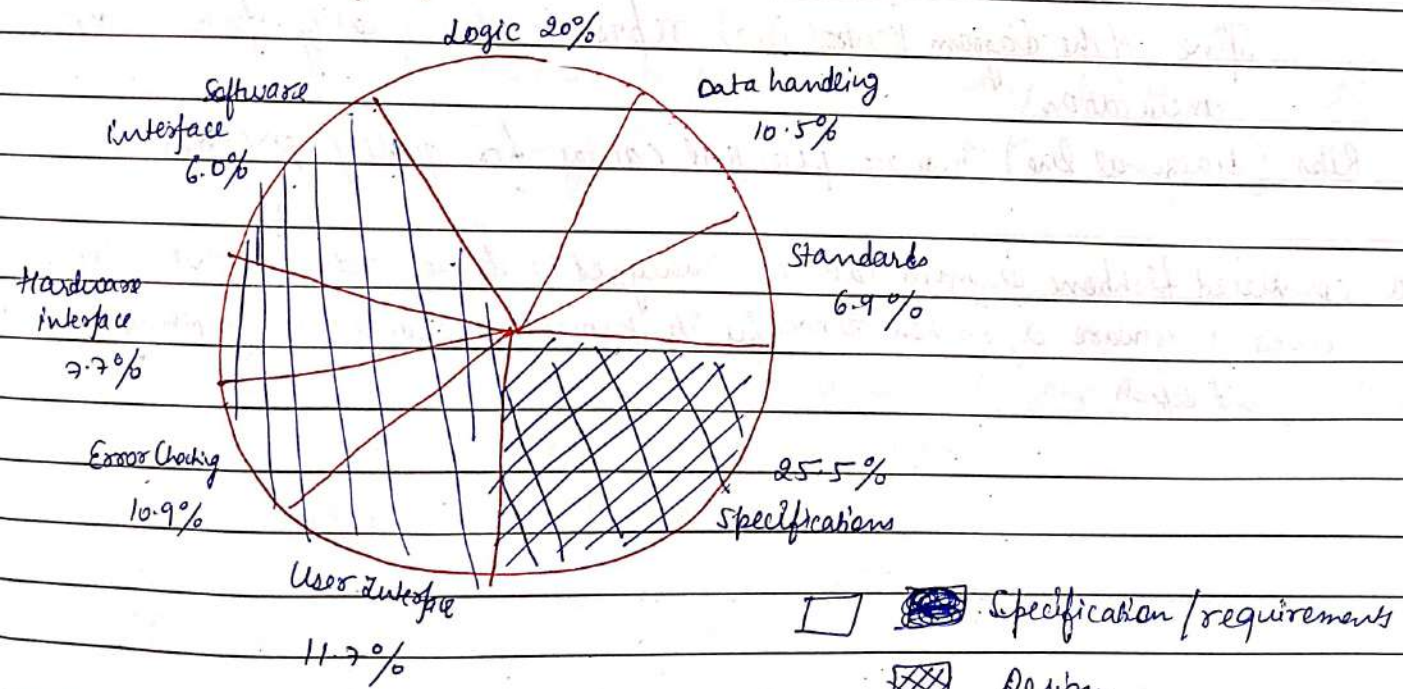
(c) The no. of errors and defects in each category is counted and ranked in descending order

(d) The overall cost of errors and defects in each category is computed

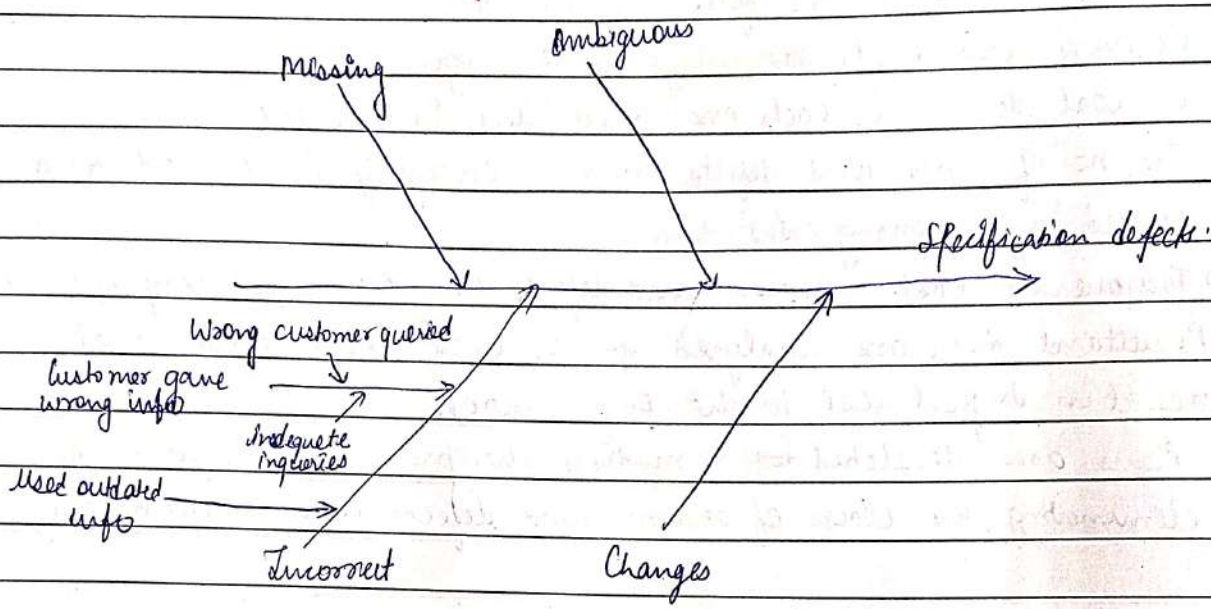
(e) Resultant data are analyzed to uncover the categories that result in highest cost to the organization

(f) Plans are developed ~~to~~ to modify the process with the intent of eliminating the class of errors and defects that is most costly

Causes of defects and their origin.



A fishbone diagram



Spine of the diagram (central line) represents the quality factor under consideration

Ribs (diagonal line) indicate potential causes for quality problem

A completed fishbone diagram can be analyzed to derive indicators that will enable a software organization to modify its process to reduce the frequency of errors and defects.

Project Metrics :

① Estimation :

(a) First use of project metrics typically occurs during project estimation.
(b) Past project metrices are used as a basis for estimating current project effort and duration.
(c) During the project, actual effort and time expended are compared with estimates to monitor progress and make necessary adjustments to the project schedule.

② Monitoring Technical work :

(a) As the project progresses and technical work begins, metric related to production rates are tracked.
(b) Errors found in each stage of software development are recorded and analyzed
(c) Technical metrics help access the quality of the design, influencing the approach to coding & testing

③ Twofold Purpose :

(a) Minimize the development schedule by adjusting overflow, identifying and avoiding delays, and mitigating risks.
(b) Improving product quality

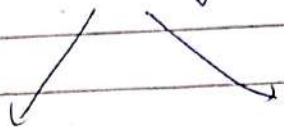
④ Model for Software Project Metrices.

(1) Inputs → resources necessary to carry out the work
(2) Outputs → The deliverables or work products generated ~~by~~ during process
(3) Results → Measures of the effectiveness and quality of outputs

This model # can be applied recursively to each phase or framework activity, where the ~~out~~ output of one activity becomes input to the next.

4.3 Software Measurement

Measurement can be categorized in two ways



direct measures Indirect measures

Direct measures of a product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.

Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability etc.

Metrics can be aggregated from individual, to team, to project and then to organizational levels

Size-Oriented Metrics

① Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the size of the software

② These metrics are based on lines of code (LOC) and other measurable aspects of the software development process. Metrics are normalized by considering the size of the software, typically measured in KLOC (thousand line of code

③ Common-size oriented metrics include:

① Error per KLOC: No. of errors detected per thousand lines of code

② Defects per KLOC: ~~Cost of development per line of code~~ No. of defects found post-release per thousand lines of code

③ $ per doc: Cost of development per line of code

④ Pages of documentation per KLOC → Amount of documentation produced for each thousand lines of code.

Project	LOC	Effort	$/doc	Pg doc	Errors	Defects	People
alpha	12100	24	168	365	134	29	5
—	—	—	—	—	—	—	—

(4) These metrics can be extended to include:
Errors per person-month, LOC per person-month, $ per page of documentation.

(5) Criticism of LOC-based metrics → Critics argue that LOC measures are programming-language-dependent and can penalize well-designed, efficient code that uses fewer lines. Additionally, predicting LOC early in a project can be ~~too~~ challenging.

Function-Oriented Metrics

(1) These metrics are based on the functionability delivered by software rather than on the size of code. The primary measure is function points (FP)

(2) Function points are derived from:
(i) User inputs → No. of distinct data inputs provided to the software.
(ii) user outputs → Application outputs such as reports, screens etc.
(iii) user inquiries → Distinct queries the software responds to
(iv) files → logical groupings of data or databases used
(v) External interfaces → Machine readable interfaces for transmitting data to other system

(3) Function points are computed using a formula that also accounts for complexity adjustments such as need for reliable backup, data communications, distributed processing

(4) ~~Extensions~~

$$FP = \text{Count total} \times [0.65 + 0.01 \times \Sigma(F_i)]$$

(Unadjusted Function Points) (Unadjus)

where count total is the sum of all FP entries obtained

(4) Extensions of Function Points:
(1) Feature Points → Used for systems with high algorithmic complexity
(2) 3D Function Point

Measurement parameter | Count | Simple | Avg | C | H | =
No. of user inputs | ☐ | × 3 | 4 | 6 | ☒ | ☐
Same for others
Count total → ☐

Comparison

LOC → easier to calculate but may not provide a complete picture, especially for complex system where algorithmic efficiency matters more than code size

Function Oriented → are language independent and focus on the functionality provided to the user, making them more applicable to diverse applications.

Calculate by formula and use it to find

Errors per FP →
Defects per FP →
Cost per FP →
FP per person-month →

~~Can~~ Do 14 questions on Pg 91 by e-commerce example with sum of rating as (f_i) ~~coeff~~ (Compemlexity Adjustment Values) $[0.65 + 0.01 \times \Sigma(F_i)]$

Read MCA 3rd sem group

Extended Function Point Metrics

① In Function Point measure, the data dimension was emphasized to the exclusion of functional & control dimension, For this reason, the function point measure was inadequate for many engineering & embedded systems.

(2) A function point extension called feature points, is a superset of the function point measure applied to software eng.

(3) The addition to the usual data dimension measure feature points include counting algorithms.

~~To calculate feats~~

(1) 3D function point

(i) Data Dimension

This dimension measures how the software handles data focuses primarily on the data inputs, outputs, inquiries, internal files and external interfaces. The counts for these data elements reflect the complexity of the system's interaction with external data sources and internal data structures

Inputs (I), Outputs (O), Inquiries (Q), Internal File (F)
External Interfaces (E)

How it's measured → The more complex the data structures, the higher the weight assigned to the data element. The complexity can be classified as low, medium or high, depending on factors like the no. of fields in an input screen or the complexity of output reports.

(ii) Functional Dimension → This dimension captures the internal processing logic and operations within the software. The function dimension refers to the number of internal operations required to convert inputs into outputs. For eg. In a payroll system, the calculations for gross pay, taxes would count as internal operation. In a control system, this could include any transformation or

processing of signals or commands to achieve the desired output.

Transformation → These are the steps involved in processing and converting input data into output data. A transformation could be as simple as a single calculation or as complex as a multi-step algorithm

How it's measured → More complex transformations and more internal operations increase the weight in this dimension

(iii) Control dimension

This dimension addresses the behavioural aspects of the system, specifically state transitions that represent changes in the system's operating mode.

Real-time systems, process control system or embedded system or embedded systems often change state based on user inputs, sensor data, or system events, making this dimension particularly important for these types of software

State Transitions → A state transition is a change from one mode of operation to another, triggered by an event. For instance, in a phone system, pressing a button might change the state from idle to dialing

How its measured → The no. of states and the complexity of transitions b/w them are counted. Systems with frequent or complex state changes would have a higher control dimension score

To compute 3D function points

Index = I + O + Q + F + E + T + R

Processing steps \ Semantic Statements	1-5	6-10	11+
1-10	Low	Low	Average
11-20	Low	Average	High
21+	Average	High	High

Complexity weighted value = $N_{il} * W_{il} + N_{ia} W_{ia} + N_{ih} W_{ih}$

N_{il}, N_{ia}, N_{ih} are the no. of occurences of element i at low, medium, high complexity

W_{il}, W_{ia}, W_{ih} are the corresponding weights for low, medium and high complexity.

4.5 Metrics for Software Quality

The goal of software engineering is to produce high-quality software systems. To achieve this, software engineers & project managers must use the metrics to measure the quality of the software throughout the development process.

① Factors that affect Quality

ⓐ Product Operation

This dimension focuses on how the software operates during its use. Key attributes include:

① Correctness → Does the software fulfill all functional requirements?

② Reliability → " " " consistently perform without failures?

③ Efficiency → does the software use resources optimally?

④ Integrity → Can the system protect itself from unauthorized access & attacks?

⑤ Usability → How easy is it for users to interact with software?

b. Product Revision

This refers to how easy it is to update or maintain the software over time. Key attribute. Include:

Maintainability
Flexibility
Testability

c. Product Transition

This refers to the software's ability to adapt to changes in its environment, including hardware & software platforms. Key attribute include

Portability
Reusability
Interoperability

(2) Measuring Software Quality

Different quality factors

(1) Correctness → It is most fundamental as it refers to whether the software meets its functional requirements

The typical metric for correctness is.

Defects per KLOC (Thousand lines of code): This metric measures the number of verified errors or bugs in the software per 1000 lines of code. Defects are counted as any instance where the software fails to conform to its specifications or requirements.

$$\frac{9+7+7+5}{6+6+8+8}$$

$$\frac{29}{24}$$

$$2\frac{3}{4}$$

$$1\frac{5}{8}$$

Importance → A high defect count signifies low correctness and usually indicates issue in requirements gathering, design

(2) Maintainability → Maintainability of reflects how easily the software can be modified after it has been deployed. Since maintenance often the most expensive phase in the software lifecycle, this is a critical measure.

Maintainability can be indirectly measured by:

(a) Mean Time to Change (MTTC): This measures the average time it takes to implement a change a request a made. MTTC includes the time taken to analyze the problem, design a solution, implement it, test the changes, and deploy them to users.

A lower MTTC suggests that the software is more maintainable, as changes can be made more quickly.

(b) Spoilage → This is a cost-oriented metric that measures the ratio of the cost to correct defects after the software has been released (spoilage) compared to the overall project cost. A high spoilage cost indicates that the system is difficult to maintain

(c) Integrity → Integrity measures the software's ability to withstand security attack, both accidental and intentional. ~~In~~.

Two key attributes are used to calculate integrity:

(1) Threat: This is the probability that an attack will occur.

(2) Security: This is the probability that system will repel or prevent the attack

Integrity formula

$$\text{Integrity} = \Sigma(1 - \text{Threat}) \times (1 - \text{Security})$$

The score is calculated by summing the results for each type of potential attack. A high integrity score indicates that the system is more secure.

④ Usability

It assesses how easy it is for users to interact with software. It can be measured by:

(a) Skill level required → The physical or intellectual skill required to learn and use the software

(b) Learning time

(c) Productivity gain

(d) User satisfaction

③ Defect Removal Efficiency

It is a quality metric that measures the effectiveness of quality assurance processes by tracking the percentage of errors removed before the software is delivered to the end-user

(a) DRE formula

$$DRE = \frac{E}{E+D}$$

E = no. of errors found before the software

D = ~~or~~ no. of ~~Errors~~ defects found after the software has been delivered to the user.

A DRE value of 1 is ideal, indicating that no defects were found after delivery. While this is often unattainable in practice, a higher

Integrity formula

$$\text{Integrity} = \Sigma(1 - \text{Threat}) \times (1 - \text{Security})$$

The score is calculated by summing the results for each type of potential attack. A high integrity score indicates that the system is more secure.

④ Usability

It assesses how easy it is for users to interact with software. It can be measured by:

(a) Skill level required → The physical or intellectual skill required to learn and use the software

(b) Learning time

(c) Productivity gain

(d) User satisfaction

③ Defect Removal Efficiency

It is a quality metric that measures the effectiveness of quality assurance processes by tracking the percentage of errors removed before the software is delivered to the end-user

(a) DRE formula

$$DRE = \frac{E}{E+D}$$

E = no. of errors found before the software

D = no. of ~~Error~~ defects found after the software has been delivered to the user.

A DRE value of 1 is ideal, indicating that no defects were found after delivery. While this is often unattainable in practice, a higher

DRE suggests that the software is of higher quality since most errors were detected and fixed before delivery

b) DRE within Development Phases

DRE can also be applied at different phases of the software development process to evaluate the effectiveness of error detection during each phas. For eg.

$$DRE_i = \frac{E_i}{E_i + E_{i+1}}$$

E_i → Errors found during software engineering activity i

E_{i+1} → Errors found in the next phase that were missed during activity i.

4.9 Establishing a software metrics Program

Step 1 : Identify your business goals

Step 2 : Identify what you want to know or learn

3 : Identify your subgoals.

4 : Identify the entities and attributes related to your subgoals

5 : Formalize your measurement goals

6 : Identify quantifiable questions and the related indicators that you will use to help you achieve your measurement goals

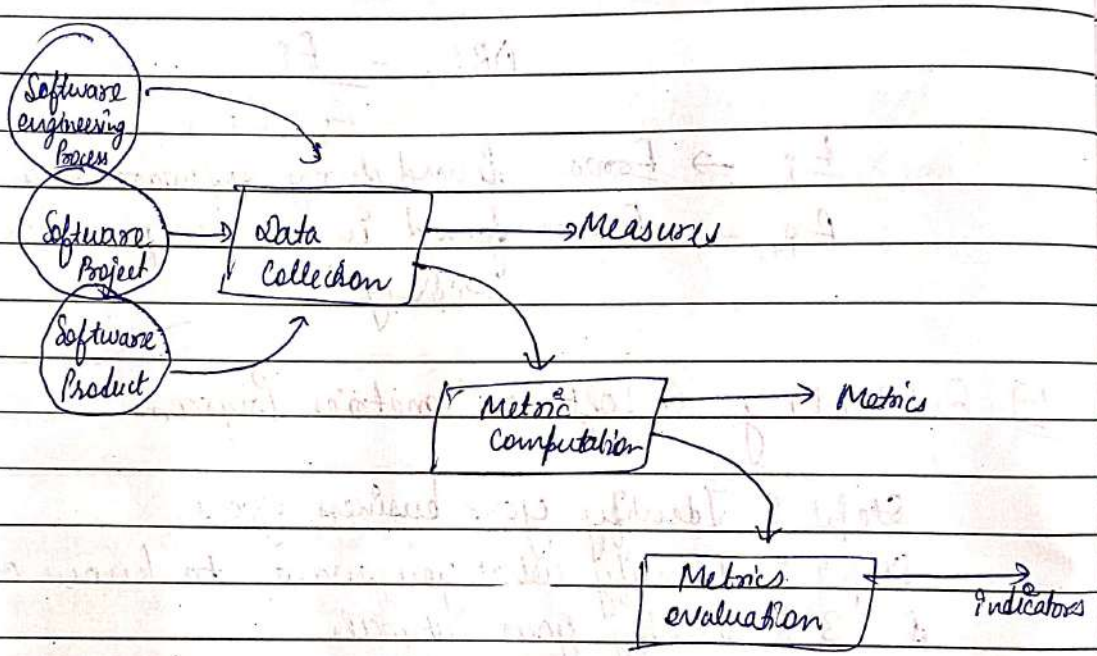
7 : Identify the data elements that you will collect to construct the indicators that help answer your questions

8 : Define the measures to be used, and make these definitions operational

9. Identify the actions that you will take to implement the measures.
10. Prepare a plan for implementing the measures.

4.7 Managing Variation : Statistical Process Control

Software metrics Collection Process



① Software Engineering Process, Software project and Software product

① These are the sources from which data is collected
② The software engineering process refers to the methodologies and procedures used during software development
③ The software project represents the actual development activities for a particular software system
④ The software product is the final outcome, the software itself.

(2) Data Collection :-

(1) Data is gathered from the process, project and product. This data is measured in specific ways to capture relevant info.

(2) The outputs of this phase are called measures, which are raw data points collected during various phases of the project (e.g. code review errors, bugs found per hour, lines of code)

(3) Metrics Computation

(1) The measures collected are then processed and computed into Metrics.

(2) Metrics are more meaningful representations of the raw data. For instance :→ no. of defects per 1,000 lines of code, or the number of errors uncovered per review hour (Er)

(4) Metrics Evaluation

(1) The computed metrics are then evaluated to derive insights or indicators.

(2) Indicators show trends, patterns, or areas of concern (e.g. an upward trend in the no. of errors may indicate declining code quality)

(3) Indicators are used to make informed decisions on improving the process

Chapter-5

(5.3) Software Scope

Software scope describes the data and control to be processed, function, performance, constraints, interfaces, and reliability. Function described in the statement of scope are evaluated and in some cases refined to provide more detail prior to the beginning of estimation. Because both cost & schedule estimates are functionally oriented,
Performance considerations encompass processing and response time requirements.
Constraints identify limits placed on the software by external hardware, available memory, or other existing systems.

(5.3.1) Obtaining Info. Necessary for Scope

The most commonly used technique to bridge the communication gap between the customer and developer and to get the communication process started is to conduct a preliminary meeting or interview.

Context-free questions lead to basic understanding of the problem

The first set of context-free questions focuses on the customer

Ex → ✓ Who is behind the request for this work?
✓ Who will use the solution?
✓ What will be the economic benefit of a successful solution?
✓ Is there another source for the solution?

The next set of questions enables the analyst to gain a better understanding of the problem.

- How would you (the customer) characterize "good" output that would be generated by a successful solution?
- What problem(s) will this solution address?
- Can you show me (or describe) the environment in which the solution will be used?
- Will any special performance issues or constraints affect the way the solution is approached?

The final set of questions focuses on the effectiveness of the meeting

- Are you the right person to answer these questions? Are answers "official"?
- Are my questions relevant to the problem that you have?
- Am I asking too many questions?
- Should I be asking you anything else?

This is the 'break the ice' approach

The Facilitated Application Specification Techniques (FAST) approach is team oriented method designed to overcome the traditional "us and them" mindset b/w customers and software engineers.

Key features

1) Joint Team formation → A joint team consisting of both customers and developers is created. This encourages cooperation rather than division, fostering a sense of shared ownership of the project.
2) Active Problem Solve & Negotiation → The team works together to identify the main problem & propose solutions

The team negotiates to resolve any differences b/w the ideal solutions from the customer's perspective and what is feasible from the technical side

(3) Preliminary requirement specification → Once problem-solving and negotiations are completed, the team outlines a preliminary set of requirements

(4) Improved communication
(open communication)

5.3.2 Feasibility

Feasibility analysis is a critical step in software planning, ensuring that the project is realistic and achievable within specific constraints.

Feasibility involves four key ~~attributes~~ dimensions ↓

(1) Technology Feasibility

- key Question → Is the project technically feasible? Is it within the current state of art

The dimension evaluates whether the required technology exists to build the project or whether it needs to be developed. The team need to ~~be~~ determine if the technical challenges can be overcome, and if defects can be reduced to an acceptable level based on the application's needs.

Risk → If the technology is immature or requires significant R&D, there may be a risk of delays or failure

2. Financial Feasibility

Key Question → Is the project financially feasible? Can it be developed within the allocated budget?

The dimension considers whether the project can be completed within a reasonable cost, balancing development costs against potential profits or the budget constraints of the organization.

Risk → If development costs exceed expectations or financial resources are limited, the project might not be feasible.

3 Time ~~Constr~~ Feasibility

Key Question → Can the project be completed within the required time frame? Will it be competitive?

Time feasibility focuses on whether the project can be completed in a timely manner, especially when there are market pressures or deadlines to meet.

Risk → If time constraints are too tight, the quality might suffer, or the project could miss its market window.

4 Resource Feasibility

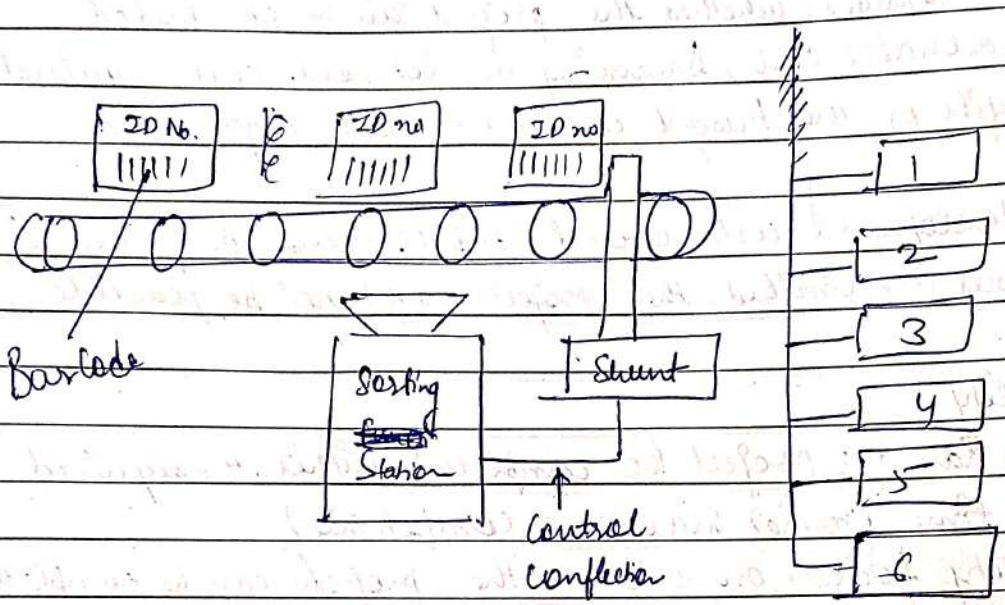
Key Question → Does the organization have the necessary resources

This dimension examines whether the organization has access to the right personnel, equipment, and tools to achieve the project's objective

Risk → If resources are scarce or the team lacks essential skills, the project might fail or encounter significant delays.

CLSS for software.

Conveyer Line Sorting System



① Software functions.

The CLSS software perform several tasks, such as :→

① Reading the barcode input from the sorting station
② Decoding the barcode into a part number.
③ Performing a database lookup to find the correct bin for each box
④ Sending control signals to the diverting mechanism to direct boxes into the correct bins
⑤ Synchronizing box positioning using input from a pulse tachometer
⑥ Maintaining a weight of box destinations for future reporting.

② Performance Requirements

① The system operates at a speed of 5 feet per minute, and the software must process each box's data before

the next box arrives at the barcode reader

② Timing constraints are critical since the software must synchronize with the conveyor line speed to ensure accurate sorting

③ System Constraints

① The software is limited by the hardware it interacts with, such as the barcode reader, sorting client, and PC. Additionally, there is a constraint on available memory.

② Boxes are evenly spaced and pass by the sorting station in random order.

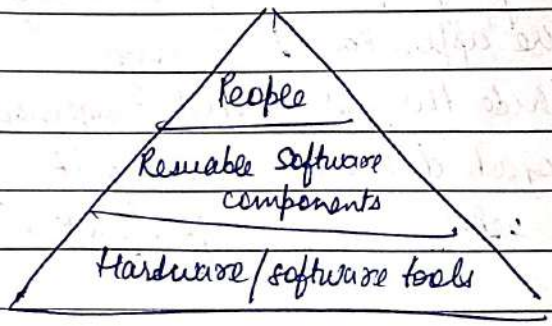
④ Impact of Changes

If the conveyor line speed increases or if the boxes are no longer evenly spaced, the complexity of the software will increase significantly, requiring more development effort

⑤ Interfaces & Reliability

5.4 Resources

Project resources



Each resource is specified with four characteristics → description of the resource, a statement of availability, time when the resource will be required, duration of time that resource will be applied.

Human Resources

① Planner identifies required roles and skills
② for small projects, one person may perform all tasks

Reusable Software Resources

① Off-the-Shelf → Pre-existing components, low-risk
② Full-experience: Familiar, low-risk components requiring minimal modifications
③ Partial-experience: Requires more modifications, higher risk.
④ New: Components built specifically for the project.

Environmental Resources

① Software engineering environment includes hardware/software tools
② Planner must ensure hardware access & availability for development / testing

Four different approaches to the sizing Problem.

① Fuzzy logic sizing → This approach uses approximate reasoning to gauge project size qualitatively. Planners identify the application type and refine its magnitude through expert judgement and historical project databases, making it ideal for projects with uncertain parameters.

② Function Point sizing → This method estimates project size based on function points, which represent user requirements, features, and the system's functionalities. Planners analyze inputs, outputs, user interactions, and data files, making it suitable for measuring complex software system.

3. Standard Component sizing → Planners identify common components in similar applications and estimate their occurrences. Using historical data, they determine the typical size for each component, facilitating a structured estimation process for software systems.

4. Change sizing → This method is essential when existing software requires modifications. Planners estimate the necessary changes and use an "effort ratio" for each type of modification.

The Software Equation

The software equation is a dynamic multivariable model designed to estimate effort and time for software development projects. This model is derived from data collected from over 4,000 contemporary software projects and relies on several variables to generate estimates.

Software Effort Equation

$$E = \left[\frac{LOC}{B^{0.333} / P}\right]^3 \times \left(\frac{1}{t^4}\right)$$ Where :

E : effort (in person-months or person-years)
LOC : lines of code, indicating project size
B : "special skills factor"
P : Productivity parameter, reflecting various factors like :
- Process maturity and management
- Software engineering practices
- Programming language sophistication
- Software environment

t : Project duration (in months or years)

Explanation of Parameters

① B (Special skills factor): This accounts for integration, testing, quality assurance, documentation. For smaller programs (5-15 KLOC) B is approximately 0.16. For larger projects exceeding 70 KLOC, B approaches 0.39. This factor grows with project complexity.

② P (productivity Parameter): Values of for P vary depending on the type of software being developed:

Real-time embedded system: P ≅ 2,000

Telecomm and system software: P ≅ 10,000

Business systems: P ≅ 28,000

Minimum Development Time & Effort:

minimum Development Time (t_{min}):

$$t_{min} = 8.14 \left(\frac{LOC}{P}\right)^{0.43}$$

(for $t_{min} > 6$ months)

Effort:

$$E = 180 \times B \times t^3$$

(for $E \geq 20$ person-months)

Do after cocomo model

Example

For CAD software with

① LOC = 33,200

② P = 12,000

$$t_{min} = 8.14 \left(\frac{33200}{12000}\right)^{0.43} = 12.6 \text{ months.}$$

For effort $E = 180 \times 0.28 \times (1.05)^3 = 58$ person months

(5.7) Empirical Estimation Models

A estimation model for computer based software uses empirically derived formulas to predict effort as a function of LOC or FP.

The Structure of Estimation models

A typical estimation model is derived using regression analysis on data collected from past software projects.

$$E = A + B \times (ev)^C$$

where A, B and C are empirically derived constants.
E is effort in person-months
ev is estimation variable (either LOC or FP)

Common Models

1. Walston-Felix: $E = 5.2 \times (KLOC)^{0.91}$
2. Boehm: $E = 3.2 \times (KLOC)^{1.05}$
3. Albrecht-Craffney (FP): $E = -13.39 + 0.0545 \times FP$

Model Calibration

Estimation models must be tailored to specific project environments for accuracy.

5.7.2 The Cocomo Model

The Cocomo model is for constructive cost Model

Object type	Complexity		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

Like function points, the object points is an indirect software that is computed using the counts of the number of

(i) Screens (at the user interface)
(ii) reports &
(iii) components likely to be required to build the application

~~Each~~

Complexity is a function of the number and source of the client and servers data tables that are required to generate the screen or report & no of views or sections presented as part of screen or report.

$$NOP = (\text{object points}) \times [(100 - \%\text{reuse})/100]$$

where NOP is defind as new object points.

Productivity rate

$$PROD = NOP / \text{person-month}$$

$$\text{estimated effort} = NOP / PROD$$

8.7.3 The Software Equation

The software equation is.

see back side

Creating a Decision

Make / buy decision

In software engineering, managers often face the challenge of deciding whether a build software internally or acquire it from external sources.

This "make/buy" decision is influenced by factors such as cost, time, functionality and availability of expertise.

Software Acquisition Options:

① Purchase Off-the-Shelf Software
② Acquire & Modify Existing Software Components
③ Custom-built Software

Steps in the Software Acquisition Process

① Develop Specifications → Define the functional and performance requirements of the desired software

② Estimate Internal Development Cost → Estimate the cost of developing the software in-house, including resource, time and a delivery date. This provides a benchmark to compare against the cost of acquiring software

(3) Select Candidate Applications or Components →
Identify three or four software packages or reusable components that best match your specifications.

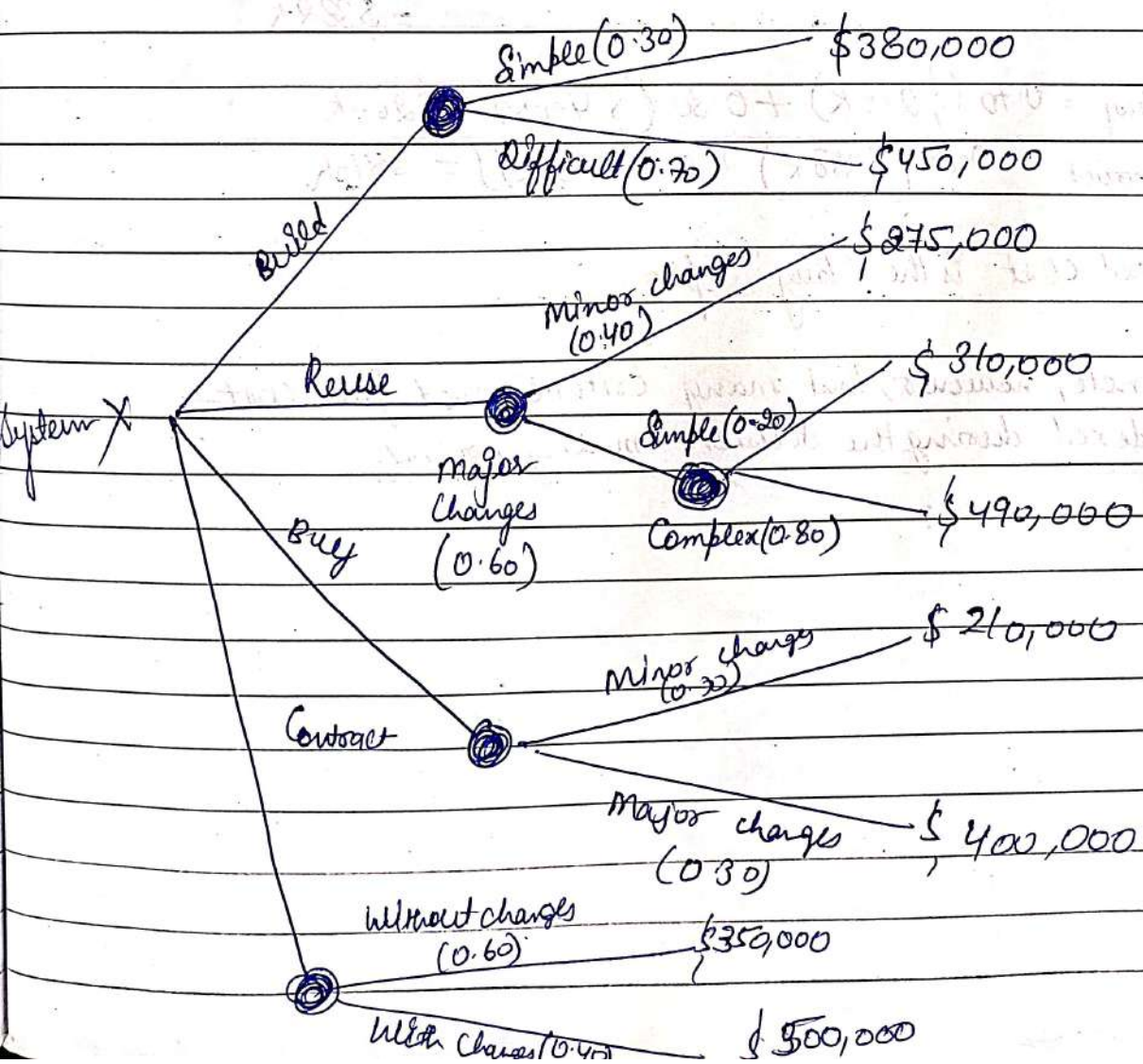
(4) Develop a Comparison Matrix : → Create a side-by-side comparison of the key functions, features and performance of the candidate software packages.

(5) Evaluate Software Based on Vendor & Product Quality

(6) Seek user feedback.

Creating a Decision Tree

The steps ~~can~~ just described can be augmented using statistical techniques ~~or~~ such as decision tree analysis.



In this case, software engineering can
(i) build system X from scratch
(ii) reuse existing "partial experience" components to construct the system
(iii) buy an available software product and modify it to meet local needs
(iv) contract the software development to an outside vendor.

The expected value for cost, computed along any branch of the decision tree, is

expected cost = Σ (path probability)$_i$ X (estimated path cost)$_i$

where i is the decision tree path. For the build path

expected cost$_{build}$ = 0.30 ($380K) + 0.70 ($450K) = $429K

expected cost$_{reuse}$ = 0.40 ($275K) + 0.60 [0.20 ($310K) + 0.80 ($490K)]
= $382K

expected cost$_{buy}$ = 0.70 ($210K) + 0.30 ($400K) = $267K

expected cost$_{contract}$ = 0.60 ($350K) + 0.40 ($500K)] = $410K

lowest expected cost is the "buy" option

It is important to note, however, that many criteria - not just cost - must be considered during the decision-making process.

~~Chat~~

3 points + fig 5.3 + fig 5.4
Process Based Est. + fig 5.5

Problem Based Estimation

Problem based estimation techniques, such as Lines of Code (LOC) and Function Points (FP), are commonly used methods in software project planning to estimate the effort, cost, time required for a project. Both techniques follow a structured approach to help project planners size the project based on measurable variables

Shared Charat of LOC & FP Estimation

1) Decomposition of software Scope
2) Estimation Variables
3) Use of historical data
4) Domain-Specific Metrics
5) Estimation - Accuracy

LOC	FP
Focus → focuses on no. of lines of codes that need to be written for each fun^n or comp.	focuses on quantifying the software's functionality
Decomposition	Complexity adjustments

3 Point or Expected Value

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

S_{opt} is the optimistic estimate
S_m is the most likely estimate
S_{pess} is the pessimistic estimate

Chapter-6

6.1 Software Risks

Risk is an inherent part of any project, and risk management is essential to ensure the successful delivery of software.

Software risks generally involve two key characteristics:

(1) Uncertainty → The risk may or may not happen. No risk is 100% certain, meaning there's always a degree of unpredictability.

(2) Loss → If the risk materializes, unwanted consequences or losses will occur. This can affect timelines, budgets, quality or overall success.

To effectively manage risks, it's essential to identify and assess them. This involves quantifying the level of uncertainty and the degree of loss.

Types of Risks

1. Project Risks → It threaten the project plan, meaning if they become real, they can lead to delays, cost overruns or resource shortages.

   Example → Budget Issues, Schedule slippage, Personnel problems, Resource Shortage, Customer related issues

   Risks in project planning are tied to the complexity, size and structural uncertainty of the project.

2. Technical Risks → Affect the quality and delivery of the software product. These risks stem from the technology, the system's architecture, and other technical aspects. Some examples include

   Design issues, Implementation problems, Verification and Testing, Maintenance issues, Obsolescene

   Technical risks often emerge because the problem is more difficult than initially anticipated, leading to challenges

In execution

3. Business Risks
   Business risks jeopardize the overall success of the software from a market or business perspective.
   These risks can affect the product's relevance or the company's strategic goals. Examples include:
   (i) ~~Market~~ Market risk: Building a product that nobody wants or needs.
   (ii) Strategic risk, Sales risk, Managment risk, Budget risk
   These risks can threaten the viability of the project or the product itself

4 known Risks → These can be identified through evaluation of the project plan, environment, and reliable information sources. For ex → Unrealistic deadlines or lack of documented requirements.

5. Predictable risks → These are risks that can be inferred from past experiences, such as staff turnover or communication problems.

6. Unpredictable risks → These are the most difficult to foresee and are often

In execution

3. Business Risks

Business risks jeopardize the overall success of the software from a market or business perspective.

These risks can affect the product's relevance or the company's strategic goals. Examples include:

(i) ~~Market~~ Market risk: Building a product that nobody wants or needs.

(ii) Strategic risk, Sales risk, Managment risk, Budget risk

These risks can threaten the viability of the project or the product itself

4 known Risks → These can be identified through evaluation of the project plan, environment, and reliable information sources. For ex → Unrealistic deadlines or lack of documented requirements.

5. Predictable risks → These are risks that can be inferred from past experience, such as staff turnover or communication problems.

6. Unpredictable risks → These are the most difficult to foresee and are often

RMMM (Risk Mitigation, Monitoring and Management)

It is a systematic approach used in software engineering to handle risks throughout the ~~dt~~ project life-cycle. The goal of RMMM is to identify potential risks, assess their impact, develop strategies to reduce or eliminate them and continously monitor them as the project progresses.

Component

(1) Risk Mitigation → Mitigation focuses on reducing the likelihood that a risk will occur or minimizing the impact if it does occur. It involves proactive steps to handle risks before they become problems.

The key achons in risk mitigation include:

(a) Identifying risks &

(b) Planning response

(2) Risk Monitoring → It involves tracking identified risks, identifying new risks, and evaluating risk reduction strategies over the course of the project.

Monitoring helps the project team understand if risks are becoming more or less likely and if the mitigation strategies are working ~~suff~~ effectively.

Key achons in risk monitoring include:

(a) Tracking risk indicators

(b) Reassessing risks regularly

(c) Documenting and reporting.

(3) Risk ~~Implementation~~ Management → It is a overall process of handling risks ensuring that the project can still ~~still~~ succeed despite uncertainties. It includes implementing contingency plans when risks become a reality and adapting the project plan to minimize the impact of those risk.

Risk management involves
(a) Executing contingency plans
(b) Adapting strategies
(c) Communication

6.3 Risk Identification

Risk Identification is a critical process in project management that seeks to specify potential threats to a project's success, such as issues related to estimates, schedules or resources. Identifying risks early allows the project manager to avoid or control them when necessary. Risks can be categorized into generic & product-specific type.

Generic risks →
Product-specific risks

Risk Item Checklist is a method for identifying risks. This checklist helps systematically evaluate potential threats under various subcategories of risks.

(1) Product size → Risks related to the overall size of software being developed or modified.

(2) Business Impact → Risks associated with management constraints or market demands.

(3) Customer Characteristics → Risks related to the customer's sophistication and the development team's ability to communicate with them effectively.

(4) Process Definition → Risks related to the degree to which a well defined and followed software development process exists within the organization

5. Development Environment → Risks associated with the quality and availability of tools needed for the project

6. Technology to be Built → Risks related to the complexity of the software and the newness of the technologies being used

7. Staff Size & Experience

6.3| ~~Assessing~~ Assessing Overall Project Risk

Questions are ordered by their relative importance

1) Have top software and customer managers formally committed to support the project?
2) Are end-users enthusiastically committed to the project and the ~~to~~ system/product to be built?
3) Are requirements fully understood by the software engineering team and their customers?
4) Have customers been involved fully in the definition of requirements?
5) Do end-users have realistic expectations?
6) Is project scope stable?
7) Does the software engineering team have the right mix of skills?
8) Does the project team have experience with the technology to be implemented?
9) Are project requirements stable?
10) Is the no. of people on the project team adequate ~~the~~ to do the job?
11) Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

6.3.2 Risk Components

The Air Force approach requires that the project manager identify the risk drivers that affect software risk components.

Performance risk → the degree of uncertainty that the product will meet its requirements and to be fit for its intended use.

Cost risk → the degree " " that the project budget will be maintained

Support risk → the degree " " " that the resultant software will be easy to correct, adapt and enhance.

Schedule risk → " " " that the project schedule will be maintained & that the product will be delivered on time.

See Fig 6.1 Pg → 178

Imp

6.4 Risk Projection

Risk projection, also called risk estimation, attempts to rate each risk in two ways — the likelihood or probability that the risk is real and the consequences of the problem associated with the risk, should it occur.

The project planner, along with other managers perform four risk projection activities: (1) establish a scale that reflects the perceived likelihood of a risk

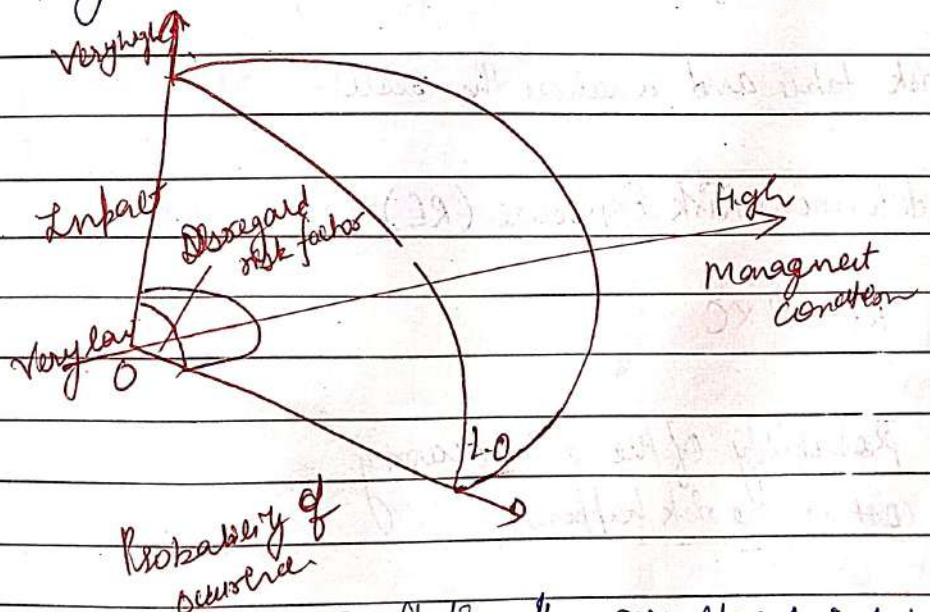
(2) delineate the consequences of the risk

(3) estimate the impact of the risk on the project and the product

(4) note the overall accuracy of the risk projection so that there will be no misunderstanding

6.4.1 Developing a Risk Table

6.4.2



The project manager studies the resultant sorted table and defines a cutoff line. The cutoff line implies that only risks that that lie above the line will be given further attention. Risks that fall below the line are re-evaluated to accomplish second-order prioritization.

6.4.2 Accessing Risk Impact

Three key factors affect the consequences of a risk when it occurs:

1) Nature → Describes the problems caused by the risk.
2) Scope → Combines the severity of the risk (how serious it is) with its overall distribution (how much of the project or how many customers are affected)
3) Timing → refers to when the impact of the risk will be felt

To assesses risk consequences, Air Force recommends ~~the steps~~ these steps:

1) Determine the average probability of occurence for each risk component
2) Assess the impact of each component using established criteria
3) Complete a risk table and analyze the results

Formula to determine Risk Exposure (RE)

$$RE = P \times C$$

where

P: probability of the risk occurring
C: cost if the risk happens

For example

Risk: 70% of planned software components are reusable
Probability (P): 80%
Impact (C): 18 components to be redeveloped at $25,200
Risk Exposure (RE): $0.80 \times \$25,200 = \$20,200$

6.5 Risk refinement

In the early stages of project planning, risks may be described broadly, but as more information becomes available, the risk can be refined into specific, manageable components. A useful way to express risks more clearly is by using the Condition-Transition-consequence format which structures the risk like this.

"Given that <condition>, there is concern that <consequence>."

General CTC Statement

"Given that all reusable software components must conform to specific design standards and that do not conform, there is concern that possibly only 70% of the planned reusable modules may be integrated into the system, resulting in the need to custom-engineer the remaining 30%

The general risk can be broken down into more specific subconditions

1) Subconditions 1: Certain reusable components were developed by a third party without knowledge of internal design standards
2) Sub " 2: Two design standard for component interface has not been finalized and may not conform to some existing reusable components.
3) " " 3: Certain reusable components are implemented in a language not supported by the target environment

6.6 Safety Risks & Hazards

Risk in software is not confined to the development phase but can extend to post deployment, particularly in cases where software failures could have significant consequences. This is especially critical in safety-critical systems like nuclear reactors, flight control system.

while the probability of failure in well-engineered systems is low, ~~but can extend to post-d~~ the potential impact of failure is severe, leading to economic damage or even loss of life. As a result, software safety and hazard are essential activities in such projects. These processes aim to identify and assess potential hazards early in development cycle. This allows for the implementation of design features that can mitigate or eliminate risk thus ensuring that the software does not compromise the overall safety of the system it controls.

This approach highlights the imp. of proactive risk management not only during development but throughout the entire software lifecycle, especially for system where failure is not an option.

Chapter-7

Basic Principles

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 should evolve from macroscopic (high-level) plans to detailed task-level schedules as the project progresses.

There are two approaches:

Fixed Deadline: The schedule is constrained by a predefined end date

Flexible Deadline: " " " developed based on the time it takes to complete tasks after careful analysis

Principles

(1) Compartmentalization → (a) Break the project into smaller, manageable tasks.

(b) Both the product and the process must be divided into parts.

(2) Interdependency → (a) ~~Identify a certain amount of effort to each task~~ Identify how tasks are related: Some are sequential, while others are happen in parallel.

(b) Understand which tasks depend on the output of others to start

(3) Time-allocation → Assign a certain amount of effort to each task, like person-days.

Set clear start and end dates for tasks, depending on task dependencies.

(4) Effort validation: → Ensure that the effort required doesn't exceed the available resources

6) Defined Outcomes.
   1) Each task should produce a specific work product.
   2) These work products often combine to into deliverables

7) Defined milestones
   1) A milestone is reached when certain tasks are completed and the work product has been reviewed & approved
   2) Milestones help track progress and ensure quality

7.2 Relationship between People & Effort

Scaling Issues in Projects

As a project grows, more people are needed, but it's not simply a matter of adding more people to speed up work. Adding too many people can disrupt the project's progress due to the complexity of communication.

The Myth of Adding Parameters

There is a common misconception that adding more programmers to a late project will help catch up. However, this often makes things worse, as new members need time to learn the system, and teaching them takes time away from the original team's work.

Communication Overhead

As team size increases, the no. of communication paths increases exponentially, leading to more overhead and lower productivity
This is demonstrated in the example :
4 engineer : 6 comm. paths ⟶ team productivity drops to 7.5%
6 engineers : 14 comm. paths ⟶ productivity continues to drop.

Example calculation

Initial team → 4 engineers can produce 20,000 LOC/year, but due to communication overhead (6 paths), they produce 18,500 LOC/year

After Adding Two more → With 6 ~~ton~~ engineers, the communication paths rise to 14, and the overall productivity drops to 18,180 LOC/year, even after adding 2 engineers capable of producing 1,680 LOC

Non-linear Productivity → The relationship b/w the no. of people and productivity is not linear adding more people does not always lead to more work getting done.

When to add People → If you need to add people to late project, make sure they are assigned to highly compartmentalized tasks that don't require much comm. with the existing team, minimizing disruption.

13 Defining a task set for the software Product

(1) The process model is populated by a set of tasks that enable a software team to define, develop, and ultimately support support computer softw

(2) A task set is a collection of ~~task~~ software engineering work tasks, milestones, and deliverables that must provide enough discipline to achieve high software quality. But at the same time, it must not burden the project team with unnecessary work.

(3) Most software organizations encounter the following projects:
1. Concept development projects that are initiated to explore some new business concept
2. New application development projects
3. Application enhancement projects
4. Application ~~bug~~ maintenance project
5. Reengineering project

9184

7.3.) Degree of Rigor

Even for a project of a particular type, the degree of rigor with which the software process is applied may vary significantly. The degree of rigor is a function of many project characteristics.

Four different degrees of rigor :-

(a) Casual → (1) Minimum task set is required
(2) Umbrella tasks like documentation are minimized but basic software eng. principles still apply
(3) Suitable for small, non-business critical projects

(b) Structured → (1) The process framework is fully applied, but activities are streamlined
(2) Umbrella tasks (Software Quality Assurance (SQA), software configuration Manag. (SCM), docum., measurement) are conducted in a streamlined manner
Suitable where quality matters

(c) Strict → full software process framework is applied with high discipline
All umbrella tasks are implemented in full.
large, complex

(d) Quick Reaction → Applies the framework, but due to an emergency only tasks essential for maintaining quality are completed
Back-filling is done after the project is delivered.
Suitable for urgent, time-sensitive situations

Adding Rigor to the Project

1) The project manager needs a systematic approach to determine the appropriate degree of rigor for a project

2) This involves defining project adaptation criteria and calculating a task set selector value to guide the decisio

7.3.2 Defining Adaptation Criteria

Adaptation criteria are used to determine the recommended degree of rigor with which the software process should be applied on a project

Eleven adaptation criteria

(1) Size of the product
(2) No of potential users
(3) Mission criticality
(4) Application longevity
(5) Stability of requirements
(6) Ease of customer/developer comm.
(7) Maturity of applicable technology.
(8) Performance constraints
(9) Embedded and non-embedded characteristics
(10) Project Staff
(11) Reengineering factors

All this are assigned a grade that range b/w 1 and 5,
1 represents a project in small subset of process tasks are required overall methodological and docu. requir are minimal

5 opposite.
complete set
substantial

Computing the Task Set Selector - An Example

Adaptation Criteria	Grade	Weight	Entry Point Multiplier: Conc.	NDev.	Enhan.	Maint.	Reeng.	Product
Size of project	2	1.2	—	1				2.4
	3	1.1	—	1				3.3
	4	1.1	—	1				
	3	0.9	—	1				
	2	~~0.7~~ 1.2	—	1				
	2	0.9		1				
	2	0.9		1				
	3	0.8		1				
	3	1.2		1				
	2	1.0		1				
	4	1.1		1				
	0	1.2		0				

All down ↑ — multiply ↑

TSS ——— : 2.8

1. Assign appropriate grades (1 to 5)
2. Assign weighting factor ranges from 0.8 to 1.2 and provides an indication of the relative importance of a particular adaptation criteria.
3. ~~Multiply~~ result of the Product of grade × weighting factor × entry point multipliers
4. Complete entries an average all entries in the product column and place result in TSS (task set selector)

TSS value helps to determine the level of process rigor a project requires

TSS < 1.2 : Casual rigor

1.0 < TSS < 3.0 : Structured rigor

TSS > 2.4 : Strict rigor

The overlapp in TSS values b/w ranges (e.g. 1.0 – 3.0 structured → 2.4 for strict)

indicates the project specific factors and experience should be used to make a final decision

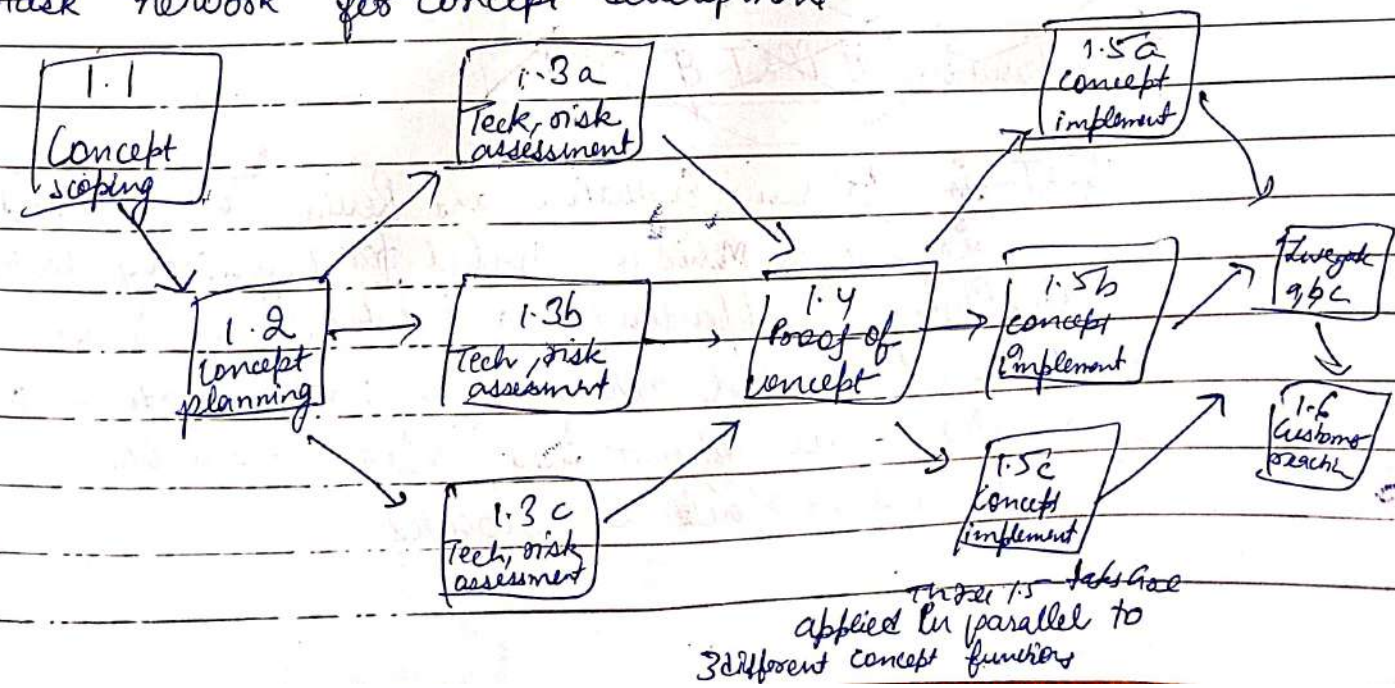
TSS = 2.8, the project falls in the overlap range b/w structured and strict rigor. The project manager can opt for a structured task set unless there is a big risk

In software engineering the task network is a graphical tool used to represent the flow of task within a project, particularly their sequence and dependencies. This tool is critical in project scheduling and coordination, especially when multiple team members are involved, and the tasks are being done in parallel

Macroscopic schedule → A task network can be simple, & high-level, focusing on the major tasks like concept development or design without diving into granular details.

Detailed Task Network → For more precise planning, the macroscopic task network is expanded into a detailed version.

A task network for concept development



Akansha

Task networks are often used as input for automated scheduling tools making it easier to adjust timelines, track progress, and manage dependencies as the project evolves.

Scheduling in software project involves applying generalized project scheduling tools and techniques, such as PERT and CPM which are commonly used in engineer projects.

Key aspects of software Project Scheduling:

(1) Estimates of Effort: Time and resources required for each task

(2) Decomposition of the Product Function: Breaking down the overall product into smaller tasks.

(3) Process Model & Task Set

(4) Task Decomposition

PERT & CPM in Software Development

PERT → Program Evaluation & Review Technique are useful for organizing tasks, identifying interdependencies, and establishing timelines.

Key Functions of PERT & CPM

PERT is Program Evaluation and Review Technique and Critical Path Method are useful for organizing tasks, identifying interdependencies, and establishing timelines. Both methods are task networks (or work breakdown structure) to define how different tasks relate to each other and how time should be allocated.

Akansha

Task networks are often used as input for automated scheduling tools. making it easier to adjust timelines, track progress, and manage dependencies as the project evolves.

Scheduling a software project involves applying generalized project scheduling tools and techniques, such as PERT and CPM which are commonly used in engineer projects.

Key aspects of Software Project Scheduling:

(1) Estimates of Effort: Time and resources required for each task

(2) Decomposition of the Product Function: Breaking down the overall product into smaller tasks.

(3) Process Model & Task Set

(4) Task Decomposition

PERT & CPM in Software Development

PERT → Program Evaluation & Review Technique) are useful for organizing tasks, identifying interdependencies, and establishing timelines.

Key Functions of PERT & CPM

PERT is Program Evaluation and Review Technique and Critical Path Method are useful for organizing tasks, identifying interdependencies, and establishing timelines. Both methods are task networks (or work breakdown structures) to define how different tasks relate to each other and how time should be allocated.

Key functions of PERT & CAM

1. Determine the Critical Path

The critical path is the chain of tasks that defines the project's total duration. If any task on this path is delayed, the whole project is delayed.

2. Most Likely Time Estimates

PERT allows for the calculation of "most likely" time estimates for individual tasks using statistical methodels

3. Calculate Boundary Times

These include

→ Earlist start

→ Latest start

→ Earlist finish

→ Latest finish

→ Total float

Importance of Boundary Time Calculations :

Calculating boundary times help project managers predict the flexibility in a schedule. For instance, if there is a delay in the design of one one module.

Tracking the Scheduling

The . Proper tracking and control ensures that the project remains on schedule and within budget, ~~and it helps~~

Key Techniques for Tracking Progress

(1) Periodic Project status meetings.

(2) Reviewing outcomes of formal Reviews.

(3) Monitoring Milestones → By comparing whether major project milestones have been achieved on time, the manager track overall progress

(4) Actual vs Planned Dates →

5. Informal meetings
6. Earned Value Analysis.

Control measures

When things are progressing smoothly, control measures are light. However, when problems arise - such as delays or resource issues - the project manager needs to intervene more directly. This might involve

Redeploying staff to high-priority areas
Reassigning resources to tackle critical bottlenecks.
Redefining the schedule to account for unforeseen challenges

Time-Boxing as a Scheduling and Control Technique

Time-boxing is a part of incremental software development and involves

1) Breaking the project into increments
2) Time-Boxing each task → fixed periods for tasks
3) Proceeding with next task

Benefits

1) Prevents project Delays
2) Ensures continous progress
3) Focuses on meeting deadlines

7.8 Earned Value Analysis

Earned Value Analysis (EVA) is a quantitative technique used to assess the progress of a software project, providing objective measurements rather than relying on subjective judgements. EVA allows project managers to track both the schedule and cost performance of a project at any given point

Steps to Perform Earned Value Analysis

(1) Determine the Budgeted Cost of Work Scheduled (BCWS):

BCWS

BCWS = Sum of budgeted costs for all tasks planned to be completed by time t.

(2) Calculate the Budget at Completion (BAC)

BAC = Sum of all BCWS for all tasks in the project.

(3) Determine the Budgeted Cost of Work Performed

BCWP = sum of the budgeted costs for all completed tasks by time t.

(4) Schedule Performance Index (SPI)

$$SPI = \frac{BCWP}{BCWS}$$

A value of close to 1.0 means the project is on schedule

(5) Schedule Variance (SV):

SV = BCWP − BCWS

A positive value indicates that the project is ahead of schedule, while a negative value means it is behind.

6. Percent Scheduled for Completion:

Percent Scheduled for Completion = $\frac{BCWS}{BAC}$

7. Percent Complete

$$\phi = \frac{BCWP}{BAC}$$

8. Actual Cost of work Performed (ACWP)

ACWP = actual cost incurred for the work that has been completed

9. CPI

$$CPI = \frac{BCWP}{ACWP}$$

CPI value close to 1.0 indicates the project is on budget

10. Cost Variance (CV)

$$CV = BCWP - ACWP$$

+ve cost CV indicates a cost saving
-ve cost CV shows overspending

Ex ⟶ BCWS = 100 person-hours
BCWP = 80 person-hours.
BAC = 200 " "
ACWP = 85 " "

$SPI = \frac{80}{100} = 0.8$

$SV = 80 - 100 = -20$

Percent scheduled $= \frac{100}{200} = 50\%$

Percent complete $= \frac{80}{200} = 40\%$

$CPI = \frac{80}{85} = 0.94$

$CV = 80 - 85 = -5$

Benefits → objective tracking, Early problem identification
Informed decision-making

7.9 Error Tracking

By systematically tracking errors and defects, project managers can compare current project performance to past projects and adjust the schedule or resources accordingly.

Error Tracking Metrics:

1. Errors per requirement specification Page (Ereq)
This metric captures the no of errors found during the review of the requirements specification.
It is useful for determining the quality of the requirement documents and identifying potential issues early in the process

2. Error per Component - Design Level (Edesign):
This measures the no. of errors discovered in the design phase, specifically related to each system component - High numbers

In this metric may indicate poorly structured designs or incomplete reviews during the design process.

3. Errors per component – Code level (Ecode):
This counts the no. of errors identified at the code level.
If the value is high, it could indicate problems with coding practices poor adherence to design, or missed issues in earlier phases.

Defect Removal Efficiency (DRE):

DRE is a key metric for understanding efficiency of the quality assurance process. It measures the % of errors that were removed during reviews & testing

$$DRE = \frac{E}{E+D}$$

E → represents the error detected and removed
D → defects
A high DRE indicates an effective review and testing process, low DRE suggests that many defects are slipping

Tracking Progress with Error Metrics

They log and report errors found in each stage. These error counts, such as Ereq, Edesign, and Ecode,
Low error counts → either the team has produced exceptionally higher quality work or the review process is not rigorous enough, potentially missing

High error counts

E_{avg} across past projects is 3.6 error per req.

Ex ⟹ $E_{req}^{current} = 2.1$

will determine whether the lower value is due to excellent work or inadequate review

7.10 Project Plan

The software Project Plan is a key document that provides essential guidelines and information to ensure smooth progress through the software development lifecycle.
It serves as both as roadmap and a reference, ensuring that all stakeholders have a clear understanding of the project's scope, resources, risks, schedule & quality management.

Functions

1) Communication of Scope and Resources:
   The plan outlines the project's goals, deliverables and the resources (people, technology, budget) necessary to achieve them.

2) Risk Identification and Mitigation:
3) Cost and Schedule Definition
4) Development Approach
5) Quality & Change Management.

Audience-Specific Presentation

Internal audience → When the plan is used internally, it may present detailed breakdowns of cost estimates using various techniques.

External audiences → When shared with external parties the plan presents a summarized, reconciled cost breakdown and a higher-level view of schedule.