M. Suhasini (IT)

PART-B

Sub. code IT733PE

2) a) Explain briefly the theoretical view of waterfall Model?

Ans: The waterfall model:

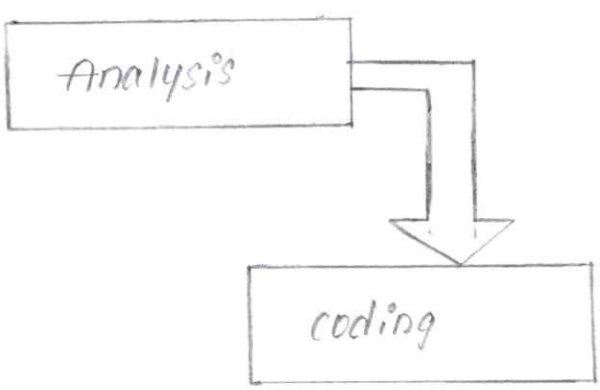
Most software engineering texts presents the waterfall model as the source of the "conventional" software process.

In Theory:

It provides an insightful and concise summary of conventional software management. Three main primary points are:

part 1: The two basic steps to building a program

There are two essential steps common to the development of computer programs: analysis and coding.

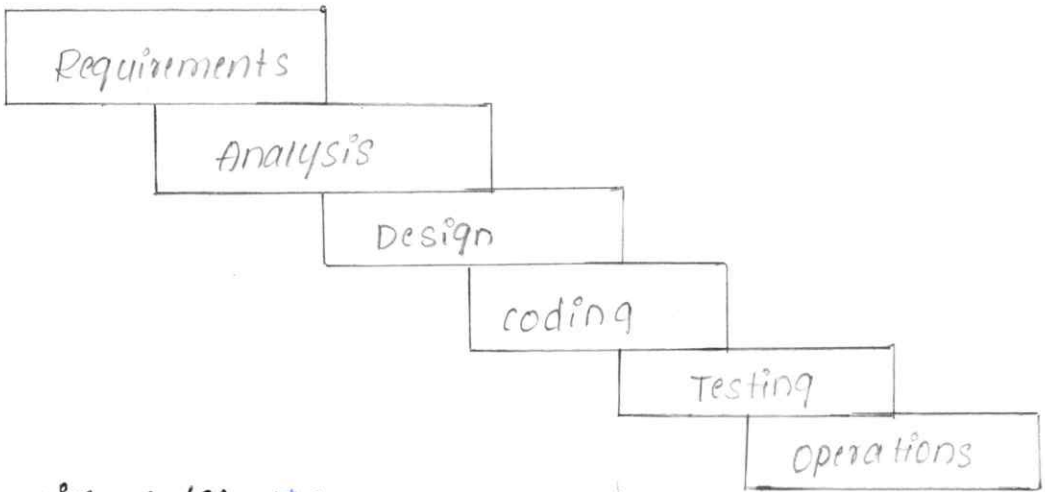


Analysis and coding both involve creative work that directly contributes to the usefulness of the end product.

1. Analysis: In this phase, the requirements are studied in detail to understand the system's functionality. Analysts create models and refine requirements to ensure they are complete, consistent and feasible.

2. coding: Developers write the actual program using suitable programming languages. It is the longest phase where design specifications are transformed into executable source code.

PART2:



1. Requirements: This phase gathers and documents what the customer needs from the software. It defines system goals, features, and expected outputs clearly before development begins.

2. Analysis: In this phase, the requirements are studied in detail to understand the system's functionality. Analysts create models and refine requirements to ensure they are complete, consistent, and feasible.

3. Design: Design converts analyzed requirements into a structural solution. It includes module design, database design to guide developers during coding.

4. coding: Developers write the actual program using suitable programming languages. It is the longest phase where design specifications are transformed into executable source code.

5. Testing: Testing ensures the software is error-free, reliable and maintaining it meets user requirements. Various tests like unit, integration system, and acceptance testing are performed.

6. Operations: The phase involves deploying the software to the end users and maintaining it throughout its life. It includes updates, bug fixes, enhancements and performance monitoring.

PART-3

five necessary improvements for waterfall model are:

1. program design comes First:

A preliminary program design phase is added before analysis to

catch issues related to storage, timing and data flow early. Designers outline processing modes, data structures, and operational constraints so major mistakes are detected before coding begins.

2. Documented the design:

Extensive documentation is needed because it communicates the system clearly to designers, manager, tester, and maintainers. In early phases, documentation is the design, and it becomes crucial for future updates and maintenance by teams not involved in development.

3. DO it twice:

The idea is to build an initial version (a prototype) first, learn from problems, and refine it into a second, improved version for real delivery. This helps identify design flaws early and ensures the final system is more reliable and stable.

4. plan, control, and monitor Testing:

Testing consumes the most time and resources, so, it must be carefully organized. A seperate test team should inspect logic paths, detect simple errors manually, and ensure final testing occurs on the actual target system.

5. Involve the customer:

customer participations is encouraged at key stages – preliminary design review, critical design reviews, and final acceptance review. This ensures early feedback, better understanding, and stong customer commitment before final delivery.

(b) Write about conventional software management performance?

Ans: Barry Boehm's "industrial software metrices TOP 10 list" is a good, objective characterization of the state of software development.

1. Finding and fixing a software problem after delivery cost 100 times more than finding and fixing the problem.
2. You can compress software development schedules 25% of nominal, nomore.
3. For every $1 you spend on development, you will spend $2 on maintenance.
4. Software development and maintenance costs are primarily a function of number of source line of code.
5. Variations among people account for the biggest differences in software productivity.
6. The overall ratio of software to hardware costs is still growing. In 1995 it was 15:85; in 1985, 85:15
7. Only about 15% of software development effort is devoted to programming.
8. Software systems and products typically cost 3 times as much per SLOC as individual software programs. Software system products (i.e., system of systems) cost 9 times as much.
9. Walkthroughs catch 60% of the errors.
10. 80% of the contribution comes from 20% of the contributors.

3) Describe how improving software process leads to better software economics and project outcomes?

Ans: Improving software processes directly enhances both the cost efficiency and the success rate of software projects. Better processes reduce unnecessary effort, improve coordination, and increases the quality of the final product. The following points explain how these improvements translate into better economics and outcomes.

1. Reduction of scrap and Rework:

A major portion of software cost comes from correcting mistakes discovered late. Improved processes help identify problems early through clear requirements, stable architecture, and continuous validation.

Benefits:
- saves time by preventing repeated rework cycles.
- leads to more predictable schedules.

2. stronger planning, control and Risk management:

Better processes establish clear policies, milestone checkpoints, and measurable indicators. This allows teams to manage risks early.

Benefits:
- prevents schedule overruns and budget blowouts.
- Improves visibility into project health.

3. Improved Team Effectiveness:

Effectiveness processes support better coordination, clearer responsibilities, and balanced team composition. They also encourage good communication and decision-making.

Benefits:
- Increases overall productivity
- Ensures efficient use of skilled resources
- Reduces confusion and unnecessary delays.

4. Enhanced Automation and Tool support:

modern processes integrate tools for requirements management, visual modeling, coding, testing, configuration management, and documentation.

Benefits:
- Automates repetitive tasks and reduces manual errors.
- speeds up development and testing activities.

5. Better and continuous Quality Assurance:

improved processes ensure that testing, evaluation and review activities occur throughout the life cycle rather than only at the end.

Benefits:
- prevents major defects from accumulating
- improves the reliability and maintainability of the product.

6. Better Alignment Between Requirements, Design, and, Implementation

improved processes ensures that all artifacts sets - requirements, design, code and deployment - remain consistent and traceable.

Benefits:
- Reduces misunderstandings and scope changes.
- ensures the product meets stakeholder needs.
- Minimizes costly redesign during integration.

4) Explain how software functionality and performance are validated during the construction phase.

Ans:- During the construction phase, validation of software functionality and performance is a central objective because this phase represents a production-oriented stage where the system must progressively mature into a usable, high-quality product. The first step in validation beings with completing all remaining components and testing them individually against their evaluation criteria. Each component undergoes unit testing to ensures its functions behave correctly in isolation, followed by integration testing where components are combined to confirm that interfaces, data flows, and interactions work as intended. This gradual integration helps detect inconsistencies early and avoids large-scale failures during final assembly.

As the system grows, the project team products alpha, beta, and intermediate test releases. These releases allow continuous assessment of functional completeness, correctness and stability. functionality is validated by executing the major use cases defined earlier in the project. Each feature is checked to ensure it satisfies the acceptance criteria of the vision, covering both normal scenarios and exception conditions. the emphasis is on minimizing scrap and rework by detecting defects early and ensuring every increment aligns with stakeholder expectations. The iterative demonstration-based validation ensures the evolving product remains consistent with requirement.

5) Discuss the challenges faced while shifting from conventional to iterative management and ways to overtome them?

Ans: challenges:

1. Teams results the cultural split from rigid phases to flexible, collaborative iterations, which can be overcome through training and mindset change. Adaption is difficult.
2. planning short iterations is difficult for teams used to long phases, and this can be solved by using user stories and clear iteration goals.
3. Iterative work demands high communication, solved by daily meetings, collaboration tools, and transparent workflows.
4. cost and schedule estimation becomes uncertain, improved by empirical methods like velocity tracking and regular estimate refinement.
5. New tools and automated testing are required, so infrastructure should be upgraded gradually.
6. stakeholders may fail to give timely feedback, addressed by fixed review cycles and clear expectations.
7. Frequent integration cause issues, reduced by automated tests and proper version-control strategies.
8. Tracking progress becomes complex, improved by using working-software metrices like velocity, completion rate and defect trends.

Improvements:

five necessary improvements for waterfall model are:

1. program design comes first: A preliminary program design phase is added before analysis to catch issues related to storage, timing, and data flow early. Designers outline processing modes, data structures, interfaces and operational constraints so major mistakes are detected before coding begins.

2. Document the design: Extensive documentation is needed because it communication the systems clearly to designers, managers, testers, and maintainers. In early phases, documentation is the design, and it becomes crucial for future updates and maintenance by teams not involved in development.

3. Do it twice: the idea is to build an initial version (a prototype) first, learn from problems, and refine it into a second, improved version for real delivery. this helps identify design flaws early and ensures the final systems is more reliable and stable.

4. plan, control and monitor testing:

Testing consumes the most time and resources, so it must be carefully organized. A seperate test team should inspect logic paths, detect simple errors manually, and ensure final testing occurs on the actual target system.

5. Involve the customer:

customer participation is encouraged at key stages - preliminary design review, critical design reviews, and final acceptance review. this ensures early feedback, better understanding, and stong customer commitment before final delivery.

6) Compare and contrast the management, engineering and programmatic artifact sets with suitable examples.

Ans:-

Aspect	Management Artifacts	engineering Artifacts	programmatic Artifacts.
purpose	plan, monitor, control the project	Describe, build & verify the system	Track execution, schedule, cost, risks.
focus Area	Business, planning, management	Technical problem + solution + implementation	project progress + resource / financial metrics.
primary Audience	stakeholders, PMs, customers.	Architects, developers, testers	PMs, Executives.
language used	Ad-hoc. text, graphics	models, code, diagrams, architecture.	Quantitative metrices, charts, timelines.
Lifecycle Role	Evolves steadily across all phases	Each phases Emphasises a different subset	used continuously for control and reporting.
Output Nature	Management decisions and agreements	Technical blueprints + software	measurable indicators for project health
Examples	Business case; SDP, WBS, release spec, change orders.	Requirements, UML, designs, source code, Executables	Schedules, milestones, charts, budget reports, risk list.

7) Explain model-Based software architecture with neat sketch

Ans:- Management perspective:

The management perspective focuses on how the architecture helps in planning, controlling, and managing a software project. It deals with communication, decision-making, risk reduction, and alignment with business goals. From a management perspective, there are all three different aspects of architecture.

1. An architecture is the design of a software system this includes all engineering necessary to specify a complete bill of materials.

2. An architecture baseline is a slice of information across the engineering artifact sets sufficient to satisfy all stakeholders that the vision can be achieved within the parameters of the business case.

3. An architecture description is an organized subset of information extracted from the design set model(s). The architecture description communication how the intangible concept is realized in the tangible artifacts.

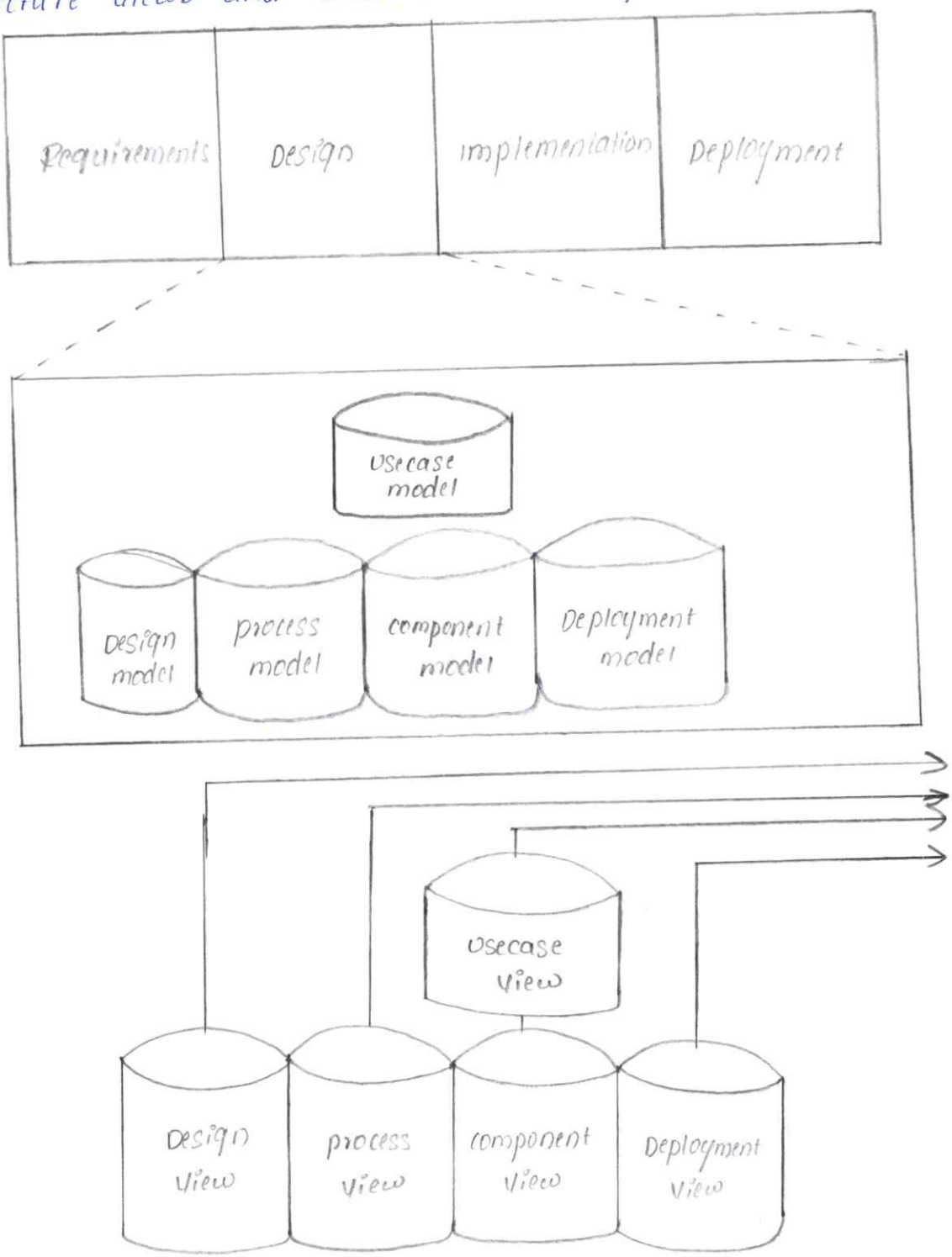
Technical perspective:

The Technical perspective of model-based software architecture explains how the architecture is represented using technical models and views. It focuses on the engineering side how the system is structured, how components interact, and how the architecture is modelled using UML.

The purposes of these views are as follows:

- Design: describes architecturally significant structures & functions of the design model
- process: describes concurrency and control used thread relationships among the design, component & deployment views.

- component: describes the structure of the implement set.
- Deployment: describes the structure of the development set summarizes the artifacts of the design set, including the architecture views and architecture description.



8) compare and contrast major and minor milestones with suitable examples.

Ans:- Major milestones: The four major milestones occur at the transistion points between life-cycle phases.

1) Life-cycle objectives milestone:
The Life-cycle objectives milestone occurs at the end of the inception phase. The goal is to present to all stakeholders a recommendation on how to proceed with development, including a plan, estimated cost and schedule, and expected benefits and cost savings. A successfully completed life-cycle objectives milestone.

2) Life-cycle Architecture milestones:
The life cycle Architecture milestones occurs at the end of the elaboration phase. The primary goal is to demonstrate an executable architecture to all stakeholders. The baseline architecture consists of both a human-readable representation and a configured-controlled set of software components captured in the engineering artifacts.

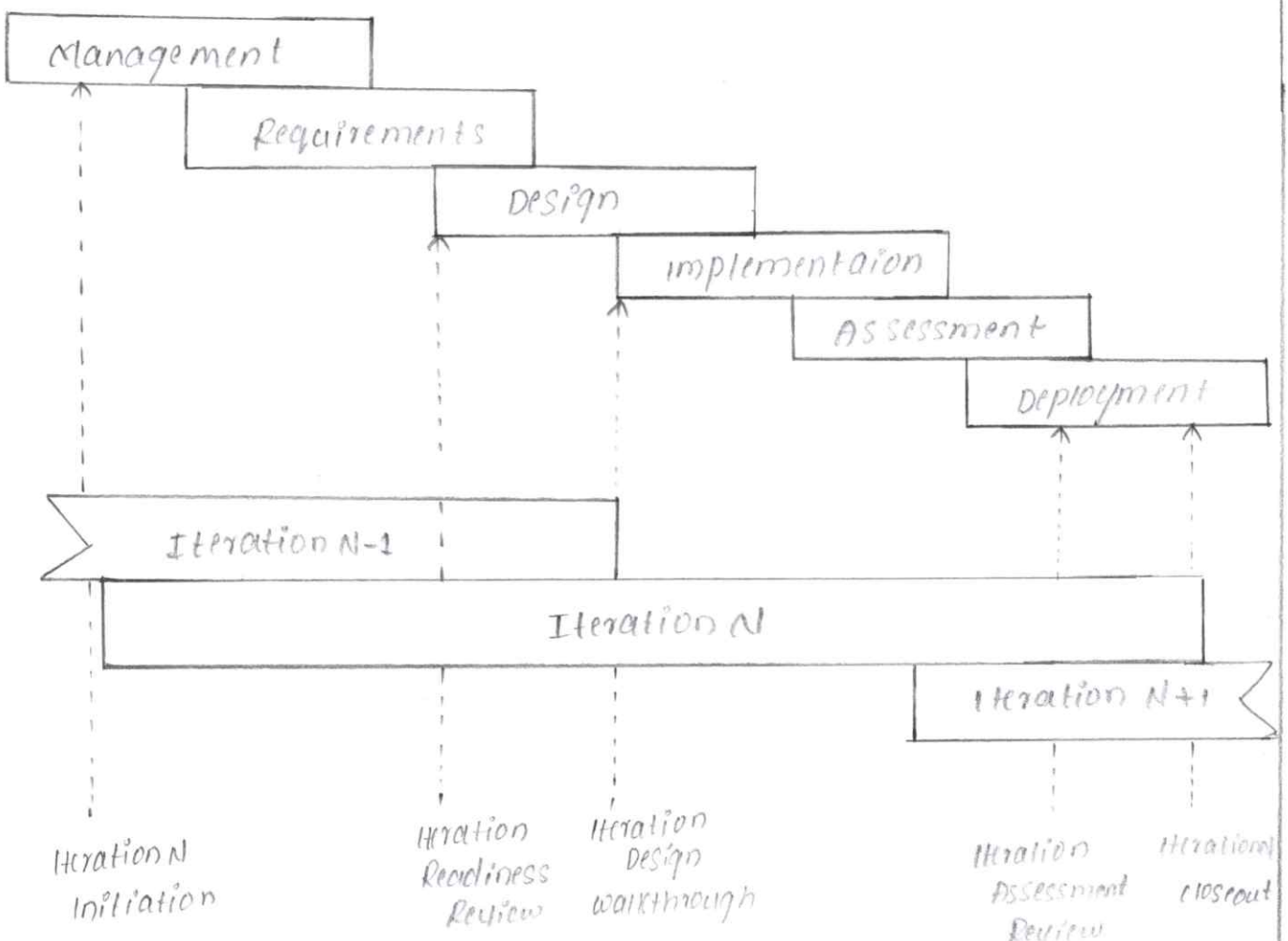
3) Initial operational capability milestone:
The initial Operational capability milestone occurs late in the construction phase. The goals are to assess the readiness of the software to begin the transistion into customer/user sites and to authorize the start of acceptance testing.

4) product Release milestone:
The product release milestone occurs at the end of the transistion phase. The goal is to access the completion of the software and its transistion to the support organization, if any. Software quality metrices are reviewed to determine whether quality is sufficient for transistion to the support organization.

Minor milestones: for most iterations, which have one-month to six-month, duration, only two minor milestones are needed: the iteration readiness review and the iteration assessment review.

1) Iteration Readiness Review: This informal milestone is conducted at the start of each iteration to review the detailed iteration plan and the evaluation criteria that have been allocated to the iteration.

2) Iteration Assessment Review:
This informal milestone is conducted at the end of each iteration to assess the degree to which the iteration achieved its objectives and satisfied its evaluation criteria, to review iteration results, to review qualification test results, to determine the amount of rework to be rework to be done, and to review the impact of the iteration results on the plan for subsequent iterations.



9) Explain the key guidelines to be followed in effective software process planning?

Ans: Key Guidelines for Effective Software process planning

Effective software process planning ensures that the entire development effort stays predictable, controllable, and aligned with stakeholder expectations. The following guidelines form the core principles of good process planning.

1. Define the scope and objectives clearly

The project scope, boundaries, goals, constraints, and expected outcomes must be clearly established at the start so that all stakeholders share a common understanding of what the system must achieve.

2. understand Requirements Before planning:

Reliable planning depends on stable, well-understood requirements. Early identification of critical use cases and essential system behaviour reduces ambiguity and lowers planning risks.

3. Adopt an architecture-first approach:

Planning must be driven by an early architectural vision. A well-defined architecture helps estimate effort, identify reusable components, assess technical risks, and determine resource allocation.

4. plan Iteratively, Not linearly

Software planning should support iterative development. Each iteration should refine requirements, evolve architecture, and reduce risks. Iterative planning improves predictability and adaptability.

5. Identify and manage Risks early:

Risk management must be integrated into the planning process. Technical, schedule, cost, requirement, and resource risks should be identified early and tracked continuously throughout the life cycle.

6. Establish Realistic schedules and milestones:
milestones must reflect achievable goals and be aligned with product increments. Each milestone should demonstrate measurable progress such as Executable architecture, tested components, or deployment readiness.

7. Allocate Resources Based on skills and Roles:
Effective planning requires assigning tasks to people with appropriate experience and strengths. Balanced teams improve productivity and reduce rework.

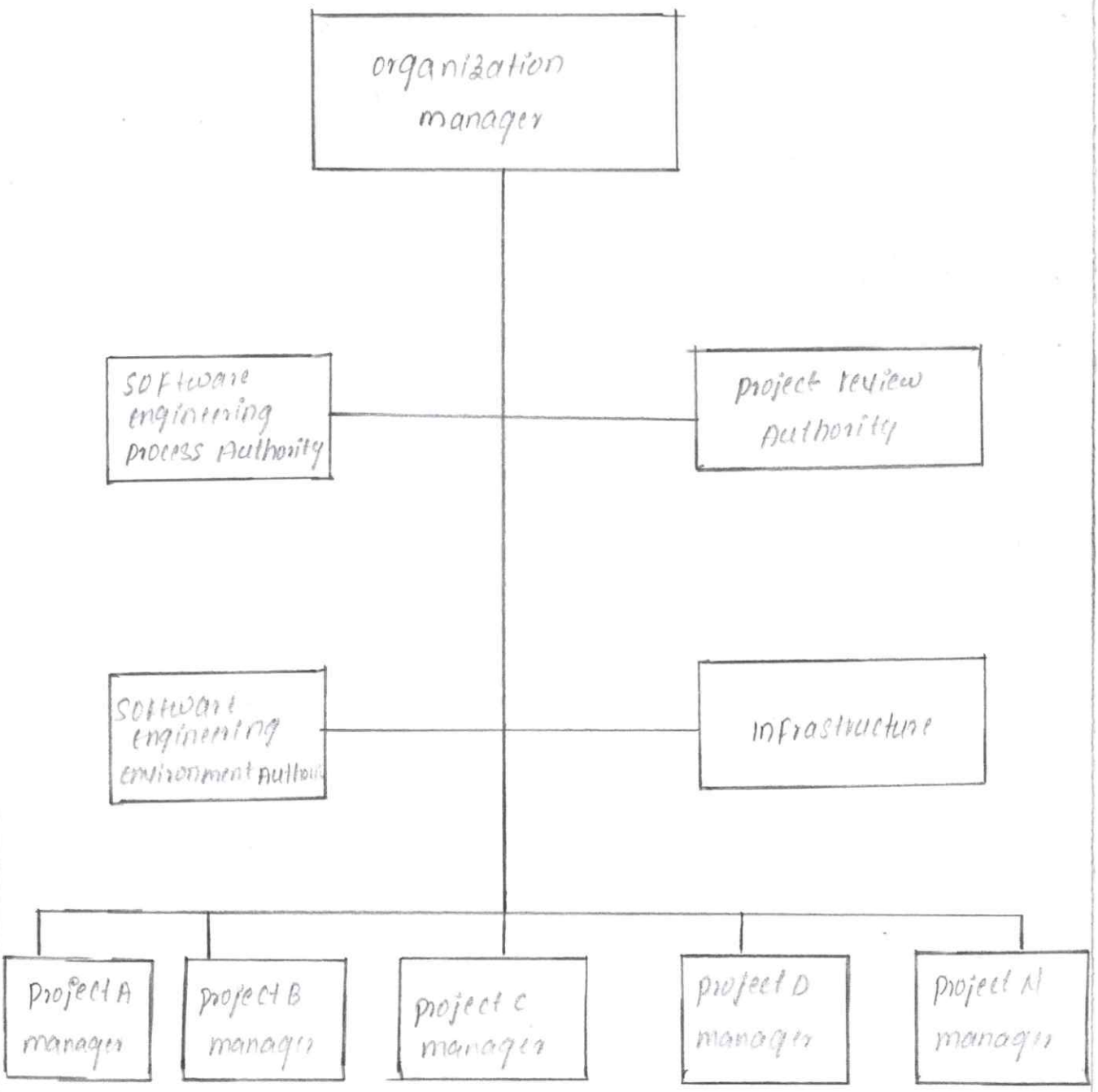
8. Define processes, TOOLS, and Environments in Advance:
The choice of development methodology, modeling tools, version control systems, testing, environments, automation tools.

9. maintain configuration and change management:
Since software evolves with continuous stakeholder feedback, planning must include mechanisms for baseline control, change tracking, version management of all artifacts.

10. Use objective metrics for monitoring progress:
measurements like defect density, earned value, iteration velocity, and milestone performance should drive decision-making.

10) What are the four components teams in a default line-of-business organization and their responsibilites?

The main features of default organization are as follows:
Responsibility for process definition & maintenance is specific to cohesive line of business. Organizational role may be fullfilled by a single individual or several different items.



1) Software engineering process Authority (SEPA):

The SEPA facilitates the exchange of information & process guidance both to and from project practitioners.

2) project Review Authority (PRA):

The PRA is the single individual responsible for ensuring that a s/w project compiles with all organizational and business unit software policies, practices and standards.

3) software engineering environment Authority (SEEA):

The SEEA is responsible for automating the organization's process, maintaining the organization's standard environment, Training process to use the environment and maintaining organization-wide reusable assets.

4) Infrastructure:

An organization's infrastructure provides human resources support, project-independent research & development, and other capital software Engineering assests.

11) Describe the major components of a software project environment?

Ans: The project environment refers to the complete set of tools, processes, infrastructure, and supporting mechanisms that enable a software project to be executed effectively.

1. prototyping environment:
- used primarily during inception and elaboration phases.
- Acts as an architecture test bed to experiment with different design choices.

2. Development Environment:
- This is the most comprehensive environment supporting all workflows of software development.
- contains a full suite of tools for
  - Requirement management
  - modelling and design
  - testing

3. maintenance environment:
- used during the post-development stage after construction and transition.
- usually supports:
  - Bug fixing
  - minor enhancements.
  - Version updates
  - performance tuning.
- Efficient change management and configuration control become critical here.

1) Round -Trip engineering

2) change management

3) (a) software change orders (SCO)

(b) configuration Baseline

(c) configuration control Based (CCB)

3) Infrastructure support

4) stakeholder environment.

1.

PART-A

a) Two Benefits of peer inspections:

1. They help detect defects.
2. Improve overall product quality.

b. Two methods to reduce software product size:

1. Source line of code (SLOC)
2. Functional units.

c. Two limitations of conventional software management:

1. Lack of feedback
2. No involvement of customer.

d. How modern software management differs from conventional management:

1. modern management uses iterative, incremental development with continuous feedback and component Based Development.
2. conventional management uses sequential development like waterfall model.

e. purpose of artifacts sets in software development:

1. They organize deliverables like plans, models, code and tests to ensure consistency.
2. provide traceability and help manage progress across engineering, management and programmatic activities

f. Two technical benefits of model-based architecture:

1. Improves system understanding through visual model and abstractions.

2. Enables early analysis, simulation, and verification of system behaviour.

g. Two advantages of inter-team workflows:

1. Enhances coordination among teams by clearly defining responsibilities and dependencies.
2. Reduce integration issues through synchronized work and shared communication channels.

h. Decision made at major milestones:

1. To achieve concurrence among all stakeholders.
2. confirmation of whether the project should continue, be revised, or stopped based on progress and risks.

i. Categories of management Indicators:

1. progress indicators
2. Quality indicators
3. Risk indicators
4. Resource indicators.

j. How pragmatic software management principles improve outcomes:

1. They focus on realistic planning, early risk handling, and continuous learning to avoid failures.
2. Encourage Iterative delivery, measurable progress, and efficient use of resources, leading to better quality and predictability.

Software Project management (SPM)
Sub code: IT833PE

11/12/25