Roll no: 3

Student name: Siddhi Mahajan

Email id: siddhiavinashmahajan@gmail.com

Q1 b: Explain the Requirements model

The Requirements model acts as a bridge between the system description and the design model, providing a comprehensive view of what the software must do. It typically consists of four main elements: Scenario-based elements (like Use Cases) that describe how users interact with the system; Class-based elements (like Class Diagrams) that define the objects and their relationships; Behavioral elements (like State Diagrams) that show how the system reacts to external events; and Flow-oriented elements (like Data Flow Diagrams) that track the movement of data through the system.

Q1 c: Explain the LOC

Lines of Code (LOC) is a direct measure of software size used in software engineering to estimate project effort, cost, and productivity.

As a size-oriented metric, it is calculated by

counting the total number of lines in a delivered

program, typically excluding comments and

blank lines. While LOC is easy to calculate and

automate, it is highly dependent on the

programming language used and can penalize

shorter, more efficient code. It is often used as

an input for estimation models like COCOMO

to predict development man-months.

Q2 a: Explain Risk and its types? Explain the RMMM plan

Risk Definition: A potential problem affecting

schedule, budget, or quality.

Project Risks: Threats to the plan like budget

overruns or resource loss.

Technical Risks: Implementation hurdles like

complex algorithms.

Business Risks: Risks concerning market

viability or competitor products.

Known vs. Predictable Risks: Found in plans vs.

extrapolated from experience.

Mitigation (R): Proactive steps to reduce risk

probability.

Monitoring (M): Tracking to see if risks are

becoming likely.

Management (M): Contingency plans for real

problems.

RMMM Structure: Plan documenting risks,

probability, impact, and owners.

RMMM Goal: Ensure stability by preparing a

systematic response.

Q2 c: Explain steps in version and change control.

Baseline Creation: Reference point reviewed

and agreed upon.

Change Request: Formal request for

modification.

Impact Analysis: Evaluating effect on software

and cost.

CRB: Authority reviewing change requests

Check-out: Preventing simultaneous edits to

the same version.

Modification: Implementing approved change locally.

Quality Audit: Testing code to meet

requirements.

Check-in: Returning code as a new version.

Labeling: Unique IDs for configuration tracking.

Reporting: Documentation for an audit trail.

Q3 a: Explain the FP Estimation techniques.

Function Points (FP): Metric for functional

complexity.

External Inputs (EI): Counting user inputs

providing data.

External Outputs (EO): Counting outputs

derived for users.

External Inquiries (EQ): Counting inputs

resulting in responses.

Internal Logical Files (ILF): Data groups inside

application.

External Interface Files (EIF): Data groups

from other systems.

Complexity Weighting: Categorizing

components as simple to complex.

UFP: Summing weighted valves.

VAF: Evaluating system characteristics.

Final Calculation: Applying formula for size.

Q3 b: Explain cohesion and Coupling.

Core Principle: Designing for module strength

and independence.

Cohesion: Functional strength inside a module.

Functional Cohesion: Single task module.

Sequential Cohesion: Output used as next

input.

Temporal Cohesion: Time-based grouping.

Coupling: Dependence between separate

modules.

Data Coupling: Simple parameter passing.

Control Coupling: passing "flags" to modules.

Common Coupling: sharing global data area.

Content Coupling: Modifying internal data of

others.

Q4 b: Explain software Re-engineering in detail.

Definition: Analyzing and altering a system to

reconstitute it in a new form.

Inventory Analysis: Assessing the portfolio to

identify candidates for re-engineering.

Document Restructuring: Updating legacy

documentation to reflect current state.

Reverse Engineering: Analyzing a program to

identify components and abstractions.

Code Restructuring: Modifying source code for

readability without changing functionality.

Data Restructuring: Redesigning data

Structures for performance or security.

Forward Engineering: Using reverse-

engineered knowledge to build anew..

BPR: Aligning software Software with new

organizational goals.

Benefits: Improves maintainability and system

reliability.

Cost-Benefit Ratio: Chosen when cheaper

than building from scratch.

Q4 c: What are the different types of maintenance?

Software Maintenance: Post-delivery

modification.

Corrective Maintenance: Fixing discovered

problems.

Adaptive Maintenance: Adapting to new

environments.

Perfective Maintenance: Adding new features.

Preventive Maintenance: Fixing latent faults.

Cost Factor: Large part of lifecycle cost

Relevance: Keeps system relevant.

Bug Fixing: Primary goal of corrective type.

Performance: Perfective type improves this.

Environment: Adaptive type modifies for this.