Q1 a: Compare FTR and Walkthrough

A Formal Technical Review (FTR) is a structured software quality control activity conducted by software engineers to uncover errors in function, logic, or implementation. It is characterized by a formal meeting with a specific agenda, a review leader, and a recorder who produces a formal summary report. In contrast, a Walkthrough is an informal peer-review process where the author of the work product guides the team through the logic or code to share knowledge and identify bugs early. While FTRs focus on strict adherence to standards, Walkthroughs are often used for early-stage brainstorming.

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 the man-months required for development.

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

Risk Definition: Risk in software engineering is a potential problem that might occur, affecting the project's schedule, budget, or quality.

Project Risks: These involve threats to the project plan, such as budget overruns, resource loss, or schedule delays.

Technical Risks: These relate to implementation hurdles, such as using unproven technology, complex algorithms, or hardware

limitations.

Business Risks: These concern the viability of the product in the market, such as a competitor releasing a better product or a shift in consumer needs.

Known vs. Predictable Risks: Known risks are identified after evaluating the project plan, while predictable risks are extrapolated from past project experience.

Risk Mitigation (R): This step involves proactive measures to reduce the probability or impact of a risk before it occurs.

Risk Monitoring (M): This involves tracking the project to see if the identified risks are becoming more likely or if mitigation is working.

Risk Management (M): This focuses on contingency planning- the steps to take if the risk actually turns into a real problem.

RMMM Plan Structure: The plan documents every risk, its probability, its potential

impact, and the specific person responsible for managing it.

The Goal of RMMM: The ultimate objective is to ensure project stability by anticipating "what could go wrong" and preparing a systematic response.

Q2 b: Explain the different techniques in white box testing.

Definition: White box testing, also known as structural testing, examines the internal logic and structure of the code.

Control Flow Testing: A strategy that uses the program's control flow as a model to design test cases for every decision point.

Basis Path Testing: A technique that allows the test case designer to derive a logical complexity measure of a procedural design and use it as a guide for defining execution paths.

Cyclomatic Complexity: A software metric that provides a quantitative measure of the logical complexity of a program, defining the number of independent paths.

Condition Testing: A test case design method that exercises the logical conditions (true/false) contained in a program module.

Data Flow Testing: Selects test paths of a program according to the locations of definitions and uses of variables in the program.

Loop Testing: A technique that focuses specifically on the validity of loop

Q2 c: Explain steps in version and change control.

Baseline Creation: Establishing a reference point in the

software lifecycle that is formally reviewed and agreed upon.

Change Request: The process begins when a stakeholder submits a formal request for a modification to a configuration item.

Impact Analysis: Evaluating the request to determine how the change will affect the software, schedule, and cost.

Change Review Board (CRB): An authority that reviews the request and decides whether to approve, deny, or defer the change.

Check-out: Developers take code from a central repository to prevent others from editing the same version simultaneously.

Modification: The developer implements the approved change in a controlled local environment.

Quality Audit: Testing and reviewing the modified code to ensure it meets \requirements without introducing new bugs.

Q3 a: Explain the FP Estimation techniques.

Function Points (FP): A size-oriented metric that measures the functional complexity of the software from the user's perspective.

External Inputs (EI): Counting each unique user input that provides distinct application-oriented data or control

information.

External Outputs (EO): Counting each unique output that derived data to be sent to the user, such as reports or screens.

External Inquiries (EQ): Counting each unique online input that results in an immediate software response (search/retrieval).

Internal Logical Files (ILF): Counting each logical group of data maintained within the application boundary.

External Interface Files (EIF): Counting logical groups of data maintained by other applications that this system references.

Complexity Weighting: Categorizing each of the five components as Simple, Average, or Complex based on their internal structure.

Unadjusted Function Points (UFP): Summing the weighted valves of all components to get a raw complexity score.

Value Adjustment Factors (VAF): Evalvating 14 general system characteristics (e.g., performance, reusability) on a scale of O to 5.

Final Calculation: Applying the formula to get the final estimated size.

Q3 b: Explain cohesion and Coupling. Explain different types with detailed example.

Core Principle: Effective modular design aims for High cohesion (internal module strength) and Low Coupling (external independence).

Cohesion Definition: A measure of the functional strength of a module-how closely the elements within it are related.

Functional Cohesion: The highest level, where a module performs exactly one task, such as "Calculate Square Root"

Sequential Cohesion: Elements are grouped because the output of one process is the input to another.

Temporal Cohesion: Elements are grouped together only because they are performed at the same time, such as initialization routines.

Coupling Definition: A measure of the degree of interdependence between two separate software modules.

Data Coupling: The best form, where modules communicate by passing simple data pieces as parameters.

Control Coupling: Occurs when one module directs the logic of another by passing "flags" or control information.

Common Coupling: Two modules share the same global data area, which can lead to data integrity issues.

Content Coupling: The worst form, where one module directly modifies or refers to the internal data of another module.

Q3 c: Explain the Spiral model of software development.

Definition: An evolutionary software process model that combines the iterative nature of prototyping with the controlled aspects of the waterfall model.

Risk-Driven: Its primary distinction is that it uses risk analysis to guide the development process at every stage.

Quadrant 1: Planning: Determination of objectives, alternatives, and constraints of the current phase of the project.

Quadrant 2: Risk Analysis: Identification and resolution of technical and management risks through prototyping or simulation.

Quadrant 3: Engineering: The actual development of the next-levelproduct, including coding, testing, and integration.

Quadrant 4: Evalvation: Customer assessment of the work product and planning for the next "circuit" of the spiral.

Cumulative Cost: Represented by the distance from the center as the spiral progresses through multiple iterations.

Iterative Nature: The project moves through these four quadrants in multiple loops until the system is complete.

Flexibility: The model allows for changes and refinements to requirements at any stage of the project lifecycle.

Suitability: Best suited for large-scale, complex, and high-risk projects where requirements may evolve.

Q4 a: Explain the general format of SRS for Hospital Management system.

Introduction: Defines the purpose, scope, and specific objectives of the Hospital Management System (HMS).

General Description: Overview of the product perspective, including user classes(Doctors, Patients, Admin) and their roles.

Functional Requirements - Registration: Details for patient admission, discharge, and electronic medical record management.

Functional Requirements - Appointments: Logic for scheduling, canceling, and tracking doctor-patient consultations.

Functional Requirements - Billing: Requirements for processing payments, insurance claims, and pharmacy charges.

Non-Functional Requirements - Security: Ensuring patient data privacy through role based access control (RBAC).

Non-Functional Requirements - Availability: Requirement for

the system to be accessible 24/7 for emergency medical services.

External Interface Requirements: How the HMS interacts with hardware (scanners, printers) and other software (insurance APIs).

Database Requirements: Description of the data schema needed to store medical histories, staff logs, and patient records.

Appendices: Includes a glossary of terms and specific diagrams like Level 0 and Level 1 DFDs for hospital workflow.

Q4 b: Explain software Re-engineering in detail.

Definition: The process of analyzing and altering a software system to reconstitute it in a new form, often to improve maintainability.

Inventory Analysis: Assessing the entire portfolio of applications to identify which ones are candidates for re-engineering based on business value.

Document Restructuring: Updating or recreating the documentation for a legacy system to reflect its current state and logic.

Reverse Engineering: The process of analyzing a program to

identify its components and their interrelationships to create high-level abstractions.

Code Restructuring: Modifying the source code to make it more readable, modular, and efficient without changing its original functionality.

Data Restructuring: Redesigning the data structures or database schema to improve performance, security,or compatibility.

Forward Engineering: Using the knowledge gained from reverse engineering to build the system anew with modern technologies.

Q4 c: What are the different types of maintenance?

Software maintenance is the process of modifying a software system after delivery to correct faults or improve performance.

Corrective Maintenance: This involves reactive modification of a software product performed after delivery to correct discovered problems and bugs. Adaptive Maintenance: This includes modifying the software to keep it usable in a changed or changing environment, such as a new operating system or hardware upgrade.

Perfective Maintenance: This focuses on enhancing the software by adding new features, improving documentation, or

refining the user interface to meet new user requirements.

Preventive Maintenance: This involves modifying software to detect and correct latent faults in the software product before they become effective faults.

Maintenance is essential because it accounts for a large percentage of the total software lifecycle cost and ensures the system remains relevant to the business.