Roll no: 4

Student name: Sai Latke

Emailid: sailatke2004@gmail.com

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 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: Potential project problem.

Project Risks: Threats like schedule delays.

Technical Risks: Hurdles like unproven tech.

Business Risks: Market viability concerns.

Known/Predictable: evaluated vs extrapolated

risks.

Mitigation (R): proactive reduction steps.

Monitoring (M): tracking risk Likelihood.

Management (M): steps for real problems.

RMMM Plan: Documents risks and owners.

Goal: Systematic preparation for issues.

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

Definition: It involves a detailed examination of

the internal logic and structure of the software

code.

Control Flow Testing: A strategy using the

program's control flow as a model to design

test cases for every decision point.

Basis Path Testing: A technique allowing the

designer to derive a logical complexity measure

to define execution paths.

Cyclomatic Complexity: A metric providing a

quantitative measure of logical complexity by

defining the number of independent paths.

Condition Testing: A method that exercises the

logical conditions (true/false) contained within

a program module.

Data Flow Testing: Selects test paths

according to the locations of definitions and

uses of variables.

Loop Testing: A technique focusing specifically on the validity of loop constructs during execution.

Statement Coverage: Ensures every single line of code in the source file is executed at least once during testing.

Branch Coverage: Validates that every possible outcome from each decision point or condition is tested

Q3 b: Explain cohesion and Coupling.

Principle: Aim for High Cohesion and Low Coupling.

Cohesion: internal strength of a module. It shows how closely related the functions inside a module are.

Functional: single task module. Best type of cohesion.

Sequential: process output used as input. Tasks follow a logical order.

Temporal: grouping by time. Activities executed at the same time are grouped.

Coupling: interdependence between modules.

Lower coupling is preferred

Data: passing parameters. Safest form of

coupling.

Control: using flags. One module controls

another.

Common: sharing global data. Can cause side

effects.

Content: modifying other module data. Worst

form of coupling.

Q3 c: Explain the Spiral model of software development.

Definition: An evolutionary software process

model combining iterative prototyping with the

controlled aspects of the waterfall model.

Risk-Driven: The primary distinction is that it

uses risk analysis to guide the development

process at every stage.

Planning: This phase involves the determination

of objectives, alternatives, and constraints for

the corrent project phase.

Risk Analysis: It focuses on the identification

and resolution of technical and management risks through prototyping or simulation.

Engineering: This stage involves the actual

development of the next-level product,

including coding, testing, and integration.

Evaluation: This quadrant requires customer

assessment of the work product and planning

for the next "circuit" of the spiral.

Cost: The cumulative cost is represented by

the distance from the center as the spiral

progresses through multiple iterations.

Iterative: The project moves through these

four quadrants in multiple loops until the

system is finally complete.

Flexibility: The model allows for necessary

changes and refinements to requirements at

any stage of the project lifecycle.

Suitability: It is best suited for large-scale,

complex, and high-risk projects where

requirements may typically 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: Provides an overview of product perspective, including user classes like Doctors and their roles.

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

Functional Requirements - Appointments: Logic for scheduling, canceling, and tracking doctor-patient consultations to ensure efficient hospital operations.

Functional Requirements - Billing: Requirements for processing payments, insurance claims, and pharmacy charges within the integrated financial module.

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

Non-Functional Requirements - Availability:
Requirement for the system to be accessible
24/7 for emergency medical services without
downtime.

External Interface Requirements: How the
HMS interacts with hardware like scanners
and printers and other external software 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.

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.

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

Document Restructuring: Updating or

recreating the documentation for a legacy system to reflect its current state.

Reverse Engineering: Analyzing a program to identify its components and interrelationships to create high-level abstractions.

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