IA 2 SEN Question Bank

2 Marks Questions

1. Define Bug, Error and Defect with example.
2. Discuss the importance of WBS.
3. Explain the steps involved in SQA plan.
4. Discuss about Project scheduling & tracking with one example.
5. Write short note on Project & Process metrics.
6. Explain Re-engineering & Reverse Engineering.
7. Explain McCall's quality factors.
8. Write short note on SCM process & its benefits. Also explain SCM Repositories.
9. Differentiate FTR and Walkthrough.
10. Define Risk. What are different types of Risk. Identify any two risks associated with software project.
11. Explain Risk Assessment & Risk Projection.
12. What do you mean by test strategy?
13. What is Object based/Use case based estimation?

5 Marks Questions

1. Explain Software Testing Techniques.
2. Describe RMMM plan in detail with example.
3. Explain LOC and Function Point estimation techniques in detail.
4. Explain COCOMO in detail.
5. Write short note on COCOMO-II.

6. Explain Change Control Processing.
7. Draw Control Flow Graph & calculate Cyclometric Complexity for the given code.
8. Compute the function point value for a project with the given data such as EI, EO, EQ, ILF, EIFEI, EO, EQ, ILF, EIF and ∑fi ∑fi considering complexity adjustment values are Average.
9. Explain Earned Value Analysis in detail.
10. Calculate EAC, ETC whose AC is 15000, BAC is 22000, EV is 13000, CPI is 0.8.

---

2 Marks Answers

Q1) Define Bug, Error and Defect with example.

- **Error:** A human mistake made by a developer or tester during coding or design.
  Example: Using `=` instead of `==` in a condition.
- **Bug:** A fault in the software that causes it to behave unexpectedly during execution.
  Example: The "Login" button doesn't respond.
- **Defect:** A deviation between the expected and actual result found during testing.
  Example: The system displays the wrong calculation result.
  In short, *error* leads to a *defect*, which when executed, causes a *bug* or *failure*.

Q2) Discuss the importance of WBS.

A **Work Breakdown Structure (WBS)** divides a project into smaller, manageable tasks.
Importance:

- Helps in **accurate estimation** of cost, time, and resources.
- Enables **clear responsibility allocation** to team members.
- Provides a **structured view** of the entire project scope.
- Improves **tracking and monitoring** of progress.
- Minimizes **risk of missing tasks**.

In software projects, WBS ensures systematic planning and better control over deliverables.

Q3) Explain the steps involved in SQA plan.

A **Software Quality Assurance (SQA) Plan** ensures that the software meets predefined quality standards.
Steps:

1. **Define quality objectives** and standards to follow (e.g., ISO 9001).
2. **Identify SQA tasks and responsibilities**.
3. **Select tools and techniques** (reviews, testing methods).
4. **Plan for audits and reviews**.
5. **Define metrics and reporting mechanisms**.
6. **Establish defect tracking and corrective actions.**
7. **Plan for documentation and deliverables.**

It ensures quality is built into every phase of development.

Q4) Discuss about Project scheduling & tracking with one example.

Project scheduling involves defining tasks, dependencies, and timelines to complete a project efficiently.

Tracking monitors progress against the planned schedule.

Example: Using a **Gantt chart**, each module (e.g., UI design, coding, testing) is assigned start and end dates.

Tracking compares actual progress with the chart to identify delays or resource bottlenecks.
Together, scheduling and tracking ensure the project stays on time and within budget.

Q5) Write short note on Project & Process metrics.

- **Project Metrics:** Measure project characteristics such as effort, cost, productivity, and defect density.
  Example: Lines of code per person-month.
- **Process Metrics:** Evaluate software development processes to improve efficiency and quality.
  Example: Average time to fix defects.
  These metrics help in **quantitative assessment**, **process improvement**, and **decision-making** for future projects.

Q6) Explain Re-engineering & Reverse Engineering.

- **Re-engineering:** Modifying existing software to improve performance, maintainability, or functionality without changing its purpose.
  Example: Migrating an old desktop app to a web-based system.
- **Reverse Engineering:** Analyzing existing software to understand its design or code for documentation or recreation.
  Example: Studying legacy code to extract design models.
  Both aim to enhance understanding and extend software life.

Q7) Explain McCall's quality factors.

McCall’s model defines **software quality** in three categories:

1. **Product Operation:** Correctness, reliability, efficiency, integrity, usability.
2. **Product Revision:** Maintainability, flexibility, testability.
3. **Product Transition:** Portability, reusability, interoperability.
   These factors ensure that software meets user needs, adapts to change, and operates effectively in different environments.

Q8) Write short note on SCM process & its benefits. Also explain SCM Repositories.

Software Configuration Management (SCM) controls changes in software throughout its lifecycle.
Process: Identification → Version Control → Change Control → Auditing → Reporting.

Benefits:

- Prevents conflicts
- Maintains consistency
- Ensures traceability.

SCM Repository: A centralized storage containing all versions of software components (source code, documents, builds). It supports versioning, rollback, and team collaboration.

Q9) Differentiate FTR and Walkthrough.

Feature	Formal Technical Review (FTR)	Walkthrough
Formality	Highly formal, structured process	Informal discussion
Participants	Moderator, reviewers, author	Author, peers
Purpose	Detect design/code defects	Understand and improve code
Documentation	Detailed and Required	Minimal or Optional
Process	Formal preparation, review meeting, and follow-up	Author presents work, others provide feedback
Example	FTR for design verification	Walkthrough for code understanding

---

Q10) Define Risk What are different types of Risk? Identify any two risks associated with software project.

Risk: A potential problem that may negatively impact a project's objectives.
Types:

1. Project Risks: Threaten project schedule, resources, or scope
   - Budget overruns, schedule delays, resource unavailability
2. Technical Risks: Threaten software quality and functionality
   - Design problems, implementation difficulties, technology issues
3. Business Risks: Threaten project viability
   - Market changes, competitive products, management changes
4. Predictable Risks: Can be anticipated based on experience
   - Staff turnover, requirement changes
5. Unpredictable Risks: Cannot be anticipated
   - Natural disasters, economic downturns

Examples:

1. Delay in component delivery (Project Risk).
2. Integration failure due to incompatible APIs (Technical Risk).

Q11) Explain Risk Assessment & Risk Projection.

- **Risk Assessment:** Identifies, analyzes, and prioritizes potential risks in a project. It involves risk identification, categorization, and probability-impact analysis.
- **Risk Projection:** Estimates the likelihood and potential consequences of each risk. It helps determine risk exposure using formulas like *Risk Exposure = Probability × Impact.*

The combination of risk assessment and projection helps project managers make informed decisions about risk management strategies.

Q12) What do you mean by test strategy?

A **Test Strategy** is a high-level document that defines the overall testing approach for a project.

It includes **testing objectives**, **levels of testing**, **test design techniques**, **environments**, **tools**, and **responsibilities**.
It ensures uniform testing practices and alignment with project goals.

Components of Test Strategy:

1. Testing Approach: Defines overall methodology (manual vs automated)
2. Testing Levels: Unit, integration, system, acceptance testing
3. Testing Types: Functional, non-functional, regression testing
4. Entry/Exit Criteria: Conditions for starting and stopping testing
5. Resource Planning: Test team structure and responsibilities
6. Tools and Environment: Testing tools and test environment setup
7. Risk Analysis: Testing risks and mitigation strategies

Example: A test strategy may specify automation for regression tests and manual testing for UI validation.

Q13) What is Object based/Use case based estimation?

- **Object-Based Estimation:** Estimates effort by identifying objects (e.g., classes, modules) and assigning complexity-based effort to each.
- Analysis Classes (key classes from requirements),
- Support Classes (additional classes, GUI multiplier 2.0-3.0),
- Effort per Class (typically 15-20 person-days),
- Use Case Count (cross-checking),
- Class Complexity (inheritance, polymorphism, encapsulation considerations)
- **Use Case-Based Estimation:** Derives estimates from system use cases by evaluating their complexity (simple, average, complex) and assigning effort multipliers.
- Use Case Points (similar to function points),
- Use Case Complexity (simple, average, complex scenarios),
- Actor Complexity (interaction complexity),
- Technical/Environmental Factors (13 complexity factors),
- UUCP (sum of use case/actor weights), UCP (UUCP adjusted by complexity factors)

Both methods are **bottom-up estimation techniques** used in object-oriented software projects for accurate effort prediction.

---

5 Marks Answers

Q1) Explain Software Testing Techniques.

1. Manual testing - Involves manual inspection and testing of the software by a human tester.
2. Automated testing - Involves using software tools to automate the testing process.
3. Functional testing - Tests the functional requirements of the software to ensure they are met.
4. Non-functional testing - Tests non-functional requirements such as performance, security, and usability.
5. Unit testing - Tests individual units or components of the software to ensure they are functioning as intended.
6. Integration testing - Tests the integration of different components of the software to ensure they work together as a system.
7. System testing - Tests the complete software system to ensure it meets the specified requirements.
8. Acceptance testing - Tests the software to ensure it meets the customer's or end-user's expectations.
9. Regression testing - Tests the software after changes or modifications have been made to ensure the changes have not introduced new defects.
10. Performance testing - Tests the software to determine its performance characteristics such as speed, scalability, and stability.
11. Security testing - Tests the software to identify vulnerabilities and ensure it meets security requirements.
12. Exploratory testing - A type of testing where the tester actively explores the software to find defects, without following a specific test plan.
13. Boundary value testing - Tests the software at the boundaries of input values to identify any defects.

14. Usability testing - Tests the software to evaluate its user-friendliness and ease of use.
15. User acceptance testing (UAT) - Tests the software to determine if it meets the end-user's needs and expectations.

[OR]

1. **Black Box Testing**

- Focuses on what the system does, not how it does it.
- The tester has no knowledge of internal code or logic.

Techniques:

- Equivalence Partitioning: Inputs are divided into valid and invalid groups to reduce test cases.
- Boundary Value Analysis (BVA): Tests the values at the edge of input ranges (e.g., min, max).
- Cause-Effect Graphing: Identifies logical relationships between input causes and output effects.
- Error Guessing: Based on tester’s experience to predict common errors.

2. **White Box Testing**

- Focuses on how the system works internally.
- Requires knowledge of the program’s logic and structure.

Techniques:

- Statement Coverage: Ensures all statements in the code are executed at least once.
- Branch Coverage: Tests all decision outcomes (True/False).
- Path Coverage: Ensures all possible control paths are tested.
- Loop Testing: Validates correct loop execution and termination.

3. **Gray Box Testing**

- Combination of both Black Box and White Box approaches.
- Tester has partial knowledge of the code.

- Useful for integration testing and detecting interface defects.

Q2) Describe RMMM plan in detail with example.

The **Risk Mitigation, Monitoring, and Management (RMMM) Plan** is a systematic approach to managing identified project risks in software engineering.
Its purpose is to detail the necessary actions to address risks before they become problems, track their status, and define contingency steps.

- **Risk Mitigation:** This involves a set of proactive steps taken to reduce the probability or impact of a risk.
- For example, if the risk is "**Staff turnover due to burn-out**," the mitigation plan might include **cross-training** critical team members and implementing **overtime limits**.
- **Risk Monitoring:** This is an ongoing activity where project management tracks indicators to determine if a risk is becoming more or less likely, or if the mitigation steps are effective.
- For the staff turnover risk, monitoring involves tracking team morale metrics and individual overtime hours.
- **Risk Management (Contingency Planning):** This defines the actions to be taken *after* a risk has materialized (a crisis management plan).
- If the staff turnover risk becomes a reality and a key developer quits, the management plan (contingency) might be to **hire a temporary contractor** and immediately **re-assign the departed developer's tasks** based on the cross-training records.

The RMMM plan ensures that the project team is prepared for potential threats, thus increasing the probability of project success.

Q3) Explain LOC and Function Point estimation techniques in detail.

1. Lines of Code (LOC) Estimation:
Estimates project size by counting total lines of source code.

- Useful in **procedural languages** like C.

- Effort = (LOC / Productivity rate).
  Example: If 10,000 LOC and productivity = 500 LOC/month → Effort = 20 person-months.
  Limitations: Depends on coding style, ignores complexity, and not language-independent.

2. Function Point (FP) Estimation:
Measures software size based on *functionality* delivered to the user.
Components:

- **External Inputs (EI)**
- **External Outputs (EO)**
- **External Inquiries (EQ)**
- **Internal Logical Files (ILF)**
- **External Interface Files (EIF)**

Each component is assigned weights (Simple, Average, Complex).

```
AFP = UFP × (0.65 + (0.01 × ΣFi))
```

Where UFP = Unadjusted Function Points and ΣFi = Complexity factors (0–5).
AFP = Adjusted Function Point is independent of programming language and better for early estimation.

Q4) Explain COCOMO in detail.

COCOMO (Constructive Cost Model) is a widely used algorithmic software cost estimation model developed by Barry Boehm. It provides a procedure for estimating development effort, cost, and schedule for software projects.

COCOMO relates the effort (`E` , in person-months) to the size of the software (`KLOC` , in thousands of lines of code) using a formula:

```
E = a × (KLOC) b × M
```

where `a` and `b` are constants based on the project mode, and `M` is a multiplier for Cost Drivers (or Effort Multipliers).

COCOMO Modes (Original Model):
The original model defined three modes, reflecting increasing complexity and project team characteristics:

- Organic Mode: Relatively small, simple projects where small teams work with familiar methods.
- Semi-Detached Mode: Intermediate projects where teams have mixed experience levels, and some flexibility exists.
- Embedded Mode: Projects that operate within a tight set of hardware, software, and operational constraints (e.g., aerospace, real-time systems). These are the most complex.

Q5) Write short note on COCOMO-II.

COCOMO II is the successor to the original COCOMO model and represents a comprehensive
hierarchy of estimation models for modern software projects. Developed by Barry Boehm, it is
designed to provide more accurate estimates by applying different models at different stages of
the software development lifecycle.

Unlike the original model, COCOMO II is not a single static model. Instead, it is a set of
integrated models that address the evolutionary nature of modern software development.
The COCOMO II hierarchy consists of three distinct models:

1. Application Composition Model:
    - This model is used during the early stages of software engineering.
    - It is ideal for projects that involve prototyping user interfaces, assessing system interaction, and evaluating technology maturity before the full architecture is known. It often uses Object Points for sizing.
2. Early Design Stage Model:
    - This model is used once requirements have been stabilized and the basic software architecture has been established.

- It allows for rougher estimates of project size before the entire design is complete, often using Function Points (FP) as the sizing metric.

3. Post-Architecture-Stage Model:
    - This is the most detailed model in the hierarchy and is used during the construction phase of the software.
    - It uses a more refined estimate of software size, typically Lines of Code (LOC), along - with a set of cost drivers (similar to Intermediate COCOMO) to produce a more accurate project effort and cost estimate.

Q6) Explain Change Control Processing.

Change Control Processing (or Change Management) is a formal, defined procedure used to manage and document changes to a project's scope, deliverables, or plans once a baseline has been established. Its primary goal is to ensure that every proposed modification is properly assessed, approved, implemented, and tracked to maintain project stability and integrity.

The Typical Process Flow:

1. **Change Request (CR) Submission:** A stakeholder or team member submits a formal request detailing the proposed change, its rationale, and its impact.
2. **Change Evaluation:** The project team, or a designated **Change Control Board (CCB)**, assesses the CR. This involves analyzing the potential impact on the **cost**, **schedule**, **resources**, **quality**, and **risk** of the project.
3. **CCB Decision:** The CCB reviews the evaluation and makes a decision: **Approve**, **Reject**, or **Defer** (pending more information or later implementation).
4. **Implementation:** If approved, the project manager integrates the change into the work plan, and the development team executes the necessary work.
5. **Verification and Documentation:** The implemented change is verified against the CR. All changes—approved, rejected, or deferred—are documented in a **Change Log** or database to maintain an audit trail.

This systematic approach prevents uncontrolled 'scope creep' and ensures that changes are only implemented when their benefits outweigh the disruption.

Q7) Draw Control Flow Graph & calculate Cyclometric Complexity for the given code.

Q8) Compute the function point value for a project with the given data such as EI, EO, EQ, ILF, EIFEI, EO, EQ, ILF, EIF and ∑fi ∑fi considering complexity adjustment values are Average.

Given: All components are Average.

- EIs = 4
- EOs = 5
- EQs = 4
- ILFs = 10
- EIFs = 7

Formulas:

- EI = 24 x 4 = 96
- EO = 16 x 5 = 80
- EQ = 22 x 4 = 88
- ILF = 4 x 10 = 40
- EIF = 2 x 7 = 14'

UFP = EI + EO + EQ + IFF + EIF
UFP = 96 + 80+88 + 40 + 14
UFP = 318

VAF = 0.65 + (0.01 x ∑fi)
VAF = 0.65 + (0.01 x 52)
VAF = 1.17

FP = UFP x VAF
FP = 318 x 1.17
FP = 372

Q9) Explain Earned Value Analysis in detail.

Earned Value Analysis (EVA) is a project management technique used to measure project performance by comparing planned progress with actual progress and cost. It helps determine whether a project is ahead or behind schedule and under or over budget.

Metrics:

1. **Planned Value (PV):**
   - PV is the authorized budget assigned to scheduled work. It represents the cost of work planned to be completed by a specific date.
2. **Earned Value (EV):**
   - EV is the budgeted cost of work actually performed. It shows how much value the completed work has earned in terms of the budget.
3. **Actual Cost (AC):**
   - AC is the total cost incurred for work performed up to a specific point in time. It reflects actual expenditure regardless of planned or earned value.

Indicators:

1. Cost Variance (CV):
   - CV = EV – AC, indicating cost performance of a project. A positive CV means under budget; negative means over budget.
2. Schedule Variance (SV):
   - SV = EV – PV, representing schedule performance. A positive SV means ahead of schedule, negative means behind schedule.
3. Cost Performance Index (CPI):
   - CPI = EV / AC, showing cost efficiency of work. A CPI above 1 indicates cost-effective performance, below 1 shows inefficiency.
4. Schedule Performance Index (SPI):
   - SPI = EV / PV, measuring time efficiency. An SPI above 1 means work is progressing faster than planned; below 1 means delay.
5. Estimate at Completion (EAC):
   - EAC predicts total project cost at completion. Calculated as BAC / CPI, it helps forecast budget requirements.
6. Budget at Completion (BAC):
   - BAC is the total planned budget for the project. It serves as a reference for cost and performance evaluation.

7. Estimate to Complete (ETC):
    - ETC = EAC – AC, estimating remaining cost to finish work. It guides managers in resource and budget planning.

Q10) Calculate EAC, ETC whose AC is 15000, BAC is 22000, EV is 13000, CPI is 0.8.

Given:

- AC = 15000
- BAC = 22000
- EV = 13000
- CPI = 0.8

Formulas:

- **EAC (Estimate at Completion)** = BAC / CPI
  = 22000 / 0.8 = **27500**
- **ETC (Estimate to Complete)** = EAC – AC
  = 27500 – 15000 = **12500**

Interpretation:

- The total project is expected to cost ₹27,500 on completion.
- Additional ₹12,500 is required to finish remaining work.
  Since CPI < 1, the project is *over budget* and needs corrective action.