SEN-2024-May-PYQ

Q1. [20 Marks]

a. What are major challenges in software engineering?

b. Write about the Non-Functional Requirements for "Online Pizza Ordering system".

   - It should contain following:
     1. Performance
     2. Availability
     3. Reliability
     4. Security
     5. Maintainability

c. Explain Evolutionary process model.

d. What is the difference between bug, error and defect? Explain with example.

e. Discuss Mc-Calls Quality factors.

Q2. [10 Marks]

a. Draw UML Use Case diagram and Class Diagram for "Smart Agriculture Monitoring System".

b. Explain COCOMO Model with example.

Q3. [10 Marks]

a. Explain Function Point Cost Estimation Technique with example.

b. Draw UML Component Diagram and Deployment Diagram for "College Management System".

Q4. [10 Marks]

a. Explain Formal Technical Review (FTR) in detail.

b. Discuss the various types of design patterns.

Q5. [10 Marks]

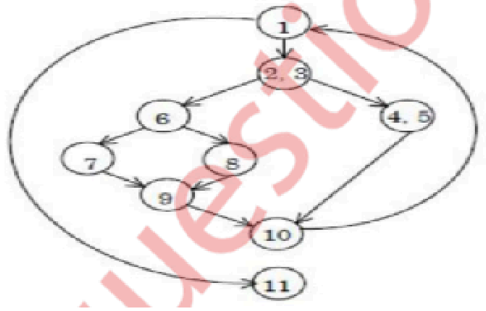
a. Explain in detail the Software Configuration Management process and benefits of SCM.

Q6. [10 Marks]

a. Explain what is a risk? Different types of risk? and describe RMMM in detail?

b) Why is cyclomatic complexity important to testers? A Given flow graph F with entry node (1) and exit node (11) is shown below. Calculate the following 10

1. How many predicate nodes are there and what are their names?
2. How many regions are there in flow graph F?
3. What is the cyclomatic complexity of flowgraph F?
4. How many nodes are there in the longest independent path?
5. How many nodes are there in flow graph F?



Q1. [5 Marks] - Answers

a. What are major challenges in software engineering?

Major Challenges in Software Engineering

Software engineering faces several technical and managerial challenges during development, deployment, and maintenance.

---

1. Quality Assurance:

- Maintaining high reliability, performance, and security standards is critical.
- Testing every possible scenario is time-consuming and costly.

2. Changing Requirements:

- User needs and market conditions often change during development.
- Adapting to these changes without disrupting progress is challenging.

3. Time and Cost Constraints:

- Projects often have tight deadlines and limited budgets.
- Balancing quality with resource limitations is difficult.

4. Team Coordination:

- Large projects require coordination among developers, testers, and clients.
- Miscommunication can lead to delays and errors.

5. Security and Privacy:

- Protecting systems from cyber threats and data breaches is a continuous challenge.

6. Technology Evolution:

- Rapid changes in tools, frameworks, and platforms require constant learning and adaptation.

b. Write about the Non-Functional Requirements for “Online Pizza Ordering system”.

Non-Functional Requirements for Online Pizza Ordering System

Non-functional requirements define **how** the system should perform rather than **what** it should do.

Below are key non-functional requirements for an **Online Pizza Ordering System**:

1. Performance:

- The system should handle **at least 500 concurrent users** without performance degradation.
- Average **page load time** should be **under 3 seconds**.
- Order placement and payment confirmation should be processed within **5 seconds**.

2. Availability:

- The system should be available **24/7**, especially during peak hours (weekends, evenings).
- Minimum **uptime of 99.5%** must be maintained.
- Backup servers should ensure service continuity during downtime.

3. Reliability:

- The system should ensure **accurate order processing** without loss or duplication.
- In case of network failure, the system must **auto-retry or queue requests**.
- Recovery from crashes should occur within **2 minutes**.

4. Security:

- All sensitive data (user credentials, payment info) must be **encrypted (SSL/TLS)**.
- Users must **authenticate** using secure login methods.
- The system should comply with **payment security standards (e.g., PCI DSS)** and prevent **unauthorized access**.

5. Maintainability:

- The system code should be **modular and well-documented** for easy updates.
- Bug fixes and feature updates should be deployable **without full system shutdown**.
- Maintenance activities should not exceed **2 hours of downtime per month**.

c. Explain Evolutionary process model.

Evolutionary Process Model

Definition:

The **Evolutionary Process Model** is a **software development approach** in which the system is developed **incrementally through multiple iterations**. Each version (or prototype) evolves based on **user feedback** until the final system is completed.

Key Characteristics:

1. **Iterative Development:**

   The software is built in small, functional parts (increments), improving with each cycle.

2. **User Feedback Driven:**

Users evaluate each version, and their feedback is used to refine requirements and features.

3. **Risk Reduction:**

   Early prototypes help identify and fix problems before full-scale development.

4. **Flexibility:**

   Suitable for projects with **changing or unclear requirements**.

5. **Early Delivery:**

   A working version of the system is available early in the development process.

Common Evolutionary Models:

- **Prototyping Model:** Builds an early working prototype for requirement clarification.
- **Spiral Model:** Combines iterative development with systematic risk analysis.

d. What is the difference between bug, error and defect? Explain with example.

Difference Between Bug, Error, and Defect

1. Error:

- **Definition:** A **mistake made by a developer** while coding or designing the software.
- **Stage:** Occurs during **development**.
- **Example:** A programmer writes `a = b + c` instead of `a = b - c`.

2. Defect:

- **Definition:** A **deviation from the expected requirement** found during testing.
- It arises due to one or more errors in the code or design.

- **Stage:** Detected by **testers** before release.
- **Example:** The “Discount Calculation” feature gives wrong total due to incorrect logic in the code.

3. Bug:

- **Definition:** A **defect accepted by the development team** after testing, meaning something is wrong in the code that needs fixing.
- **Stage:** Reported during **testing or after release**.
- **Example:** The “Add to Cart” button does not work on the product page — reported as a bug.

Term	Found By	Stage	Meaning	Example
Error	Developer	Coding/Design	Human mistake in logic or syntax	Wrong formula used
Defect	Tester	Testing	Functionality not meeting requirements	Incorrect output
Bug	Tester/User	Testing or Production	Confirmed defect in code	Button not working

e. Discuss Mc-Calls Quality factors.

McCall’s Quality Factors

Definition:

McCall’s Quality Model (proposed by Jim McCall, 1977) defines key **software quality attributes** grouped into factors that ensure a system meets user needs and performs reliably.

Main Quality Factors:

1. **Product Operation Factors** – How well the software operates:
    - **Correctness:** Software meets all specified requirements.
    - **Reliability:** Performs consistently without failure.
    - **Efficiency:** Uses system resources (CPU, memory) optimally.
    - **Integrity:** Protects against unauthorized access or data loss.
    - **Usability:** Easy for users to learn and operate.

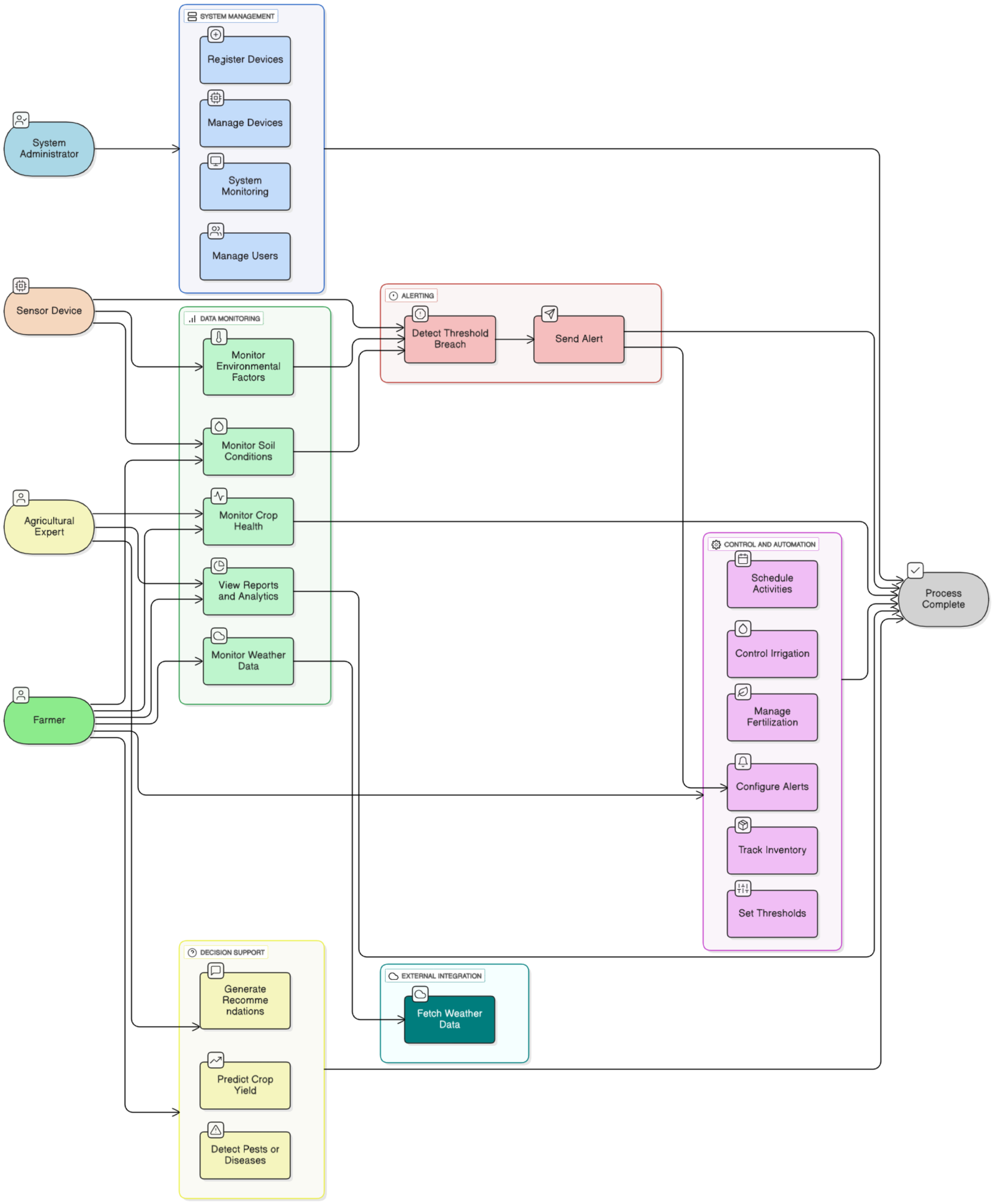
2. **Product Revision Factors** – Ease of modification and maintenance:
    - **Maintainability:** Easy to fix defects or update software.
    - **Flexibility:** Easy to modify for new features or environments.
    - **Testability:** Easy to test and verify correctness.

3. **Product Transition Factors** – Adaptability to new environments:
    - **Portability:** Can be used in different hardware or OS environments.
    - **Reusability:** Code or components can be reused in other projects.
    - **Interoperability:** Can interact with other systems or software.

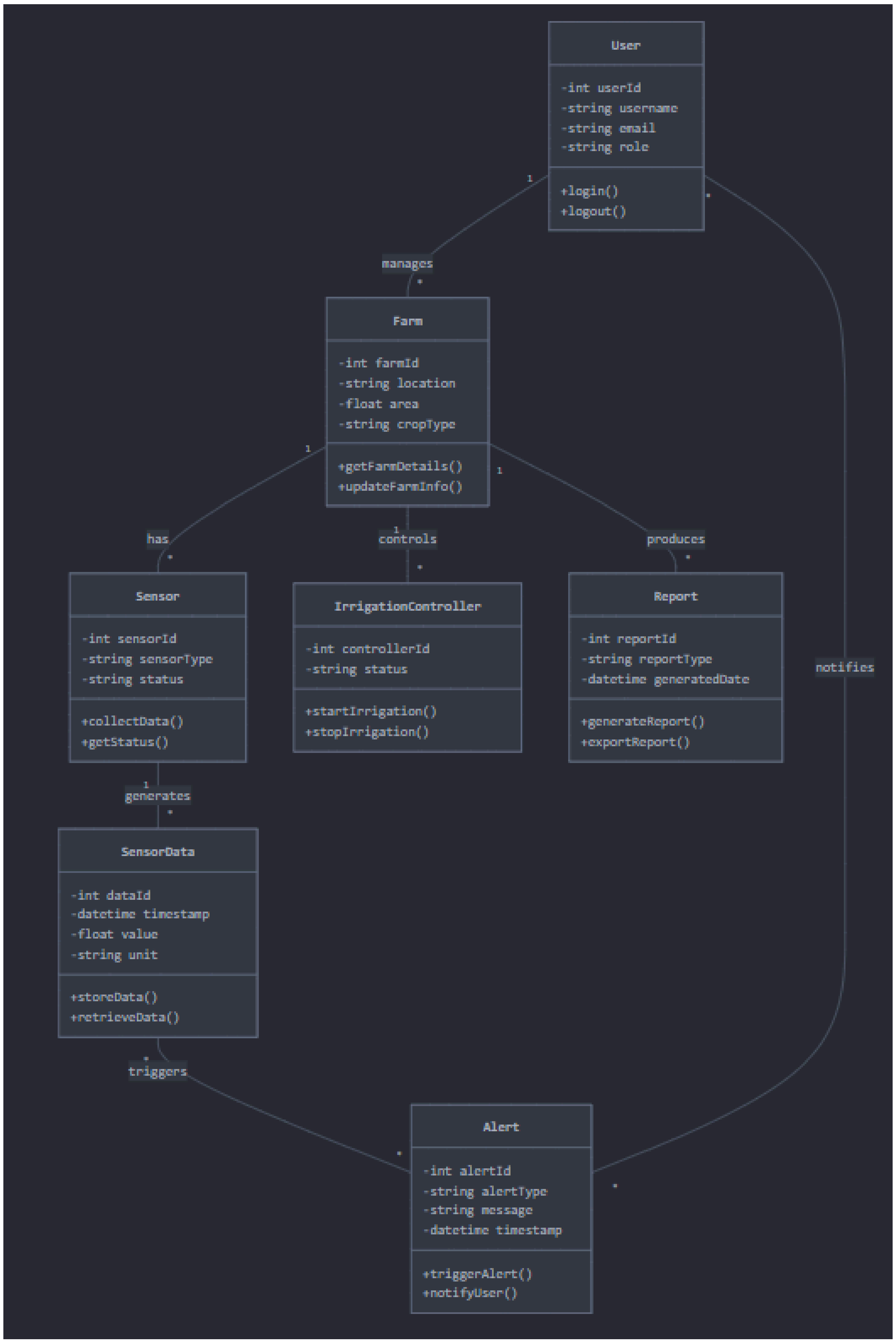
Q2. [10 Marks] - Answers

a. Draw UML Use Case diagram and Class Diagram for "Smart Agriculture Monitoring System".

UML Use Case diagram:


SYSTEM MANAGEMENT
Register Devices
Manage Devices
System Monitoring
Manage Users
System Administrator
Sensor Device
Agricultural Expert
Farmer
DATA MONITORING
Monitor Environmental Factors
Monitor Soil Conditions
Monitor Crop Health
View Reports and Analytics
Monitor Weather Data
ALERTING
Detect Threshold Breach
Send Alert
CONTROL AND AUTOMATION
Schedule Activities
Control Irrigation
Manage Fertilization
Configure Alerts
Track Inventory
Set Thresholds
Process Complete
DECISION SUPPORT
Generate Recomme ndations
Predict Crop Yield
Detect Pests or Diseases
EXTERNAL INTEGRATION
Fetch Weather Data

Class diagram:



b. Explain COCOMO Model with example.

COCOMO Model (Constructive Cost Model)

1. Introduction:

- The **COCOMO Model** is a **software cost estimation model** developed by **Barry W. Boehm (1981)**.
- It helps estimate **effort, time, and cost** required to develop software based on its **size (in KLOC – Kilo Lines of Code)**.
- The model assumes that effort increases exponentially with project size and complexity.

2. Purpose:

- To predict **person-months of effort**, **development time**, and **overall project cost**.
- Useful for **project planning, budgeting, and resource allocation**.

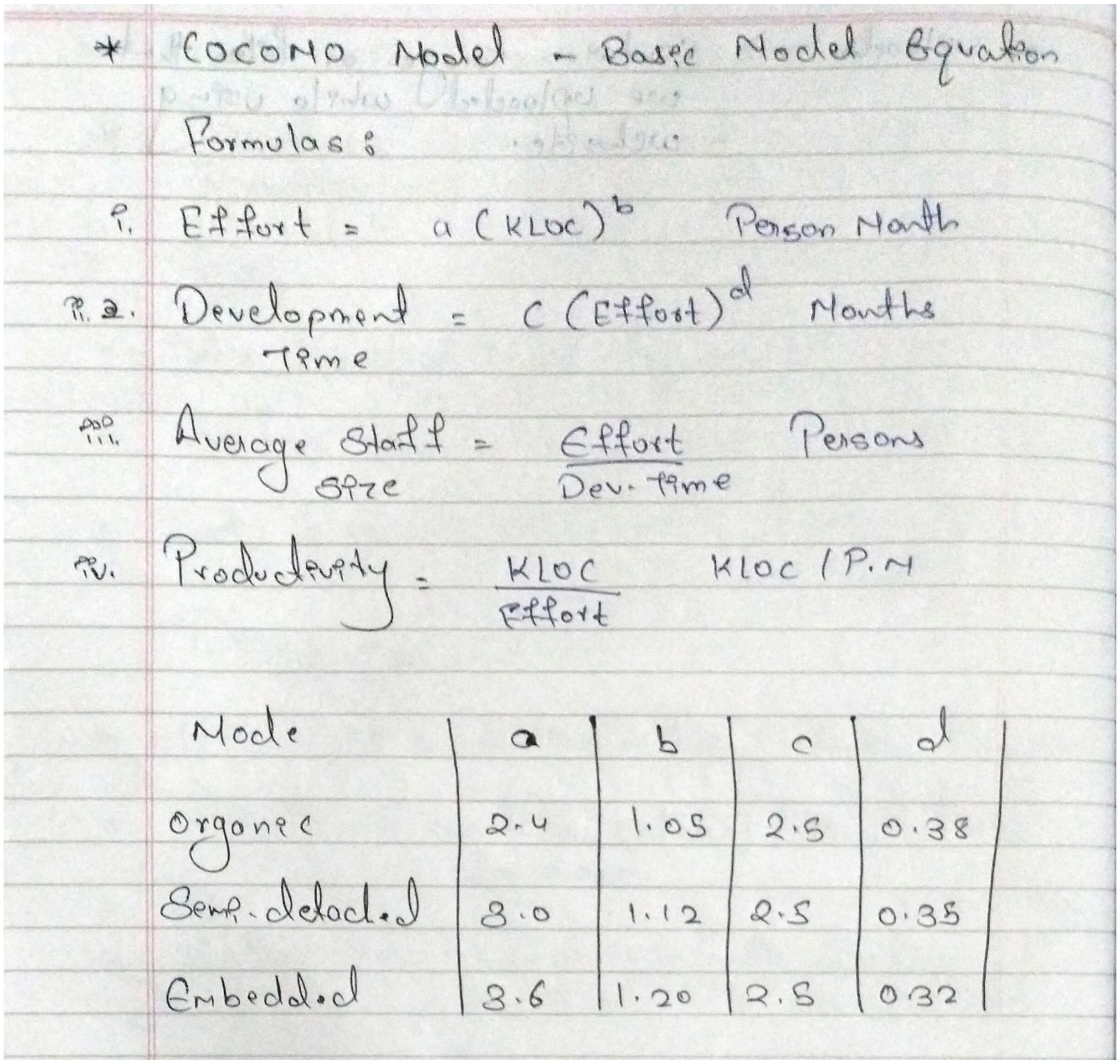
3. Types of COCOMO Models:

1. **Basic COCOMO:**

   Provides a rough estimate of effort based only on project size.
2. **Intermediate COCOMO:**

   Adds cost drivers like product reliability, experience, tools, and hardware constraints.
3. **Detailed COCOMO:**

   Includes all cost drivers and phase-wise effort distribution.



* COCOMO Model – Basic Model Equation

Formulas :

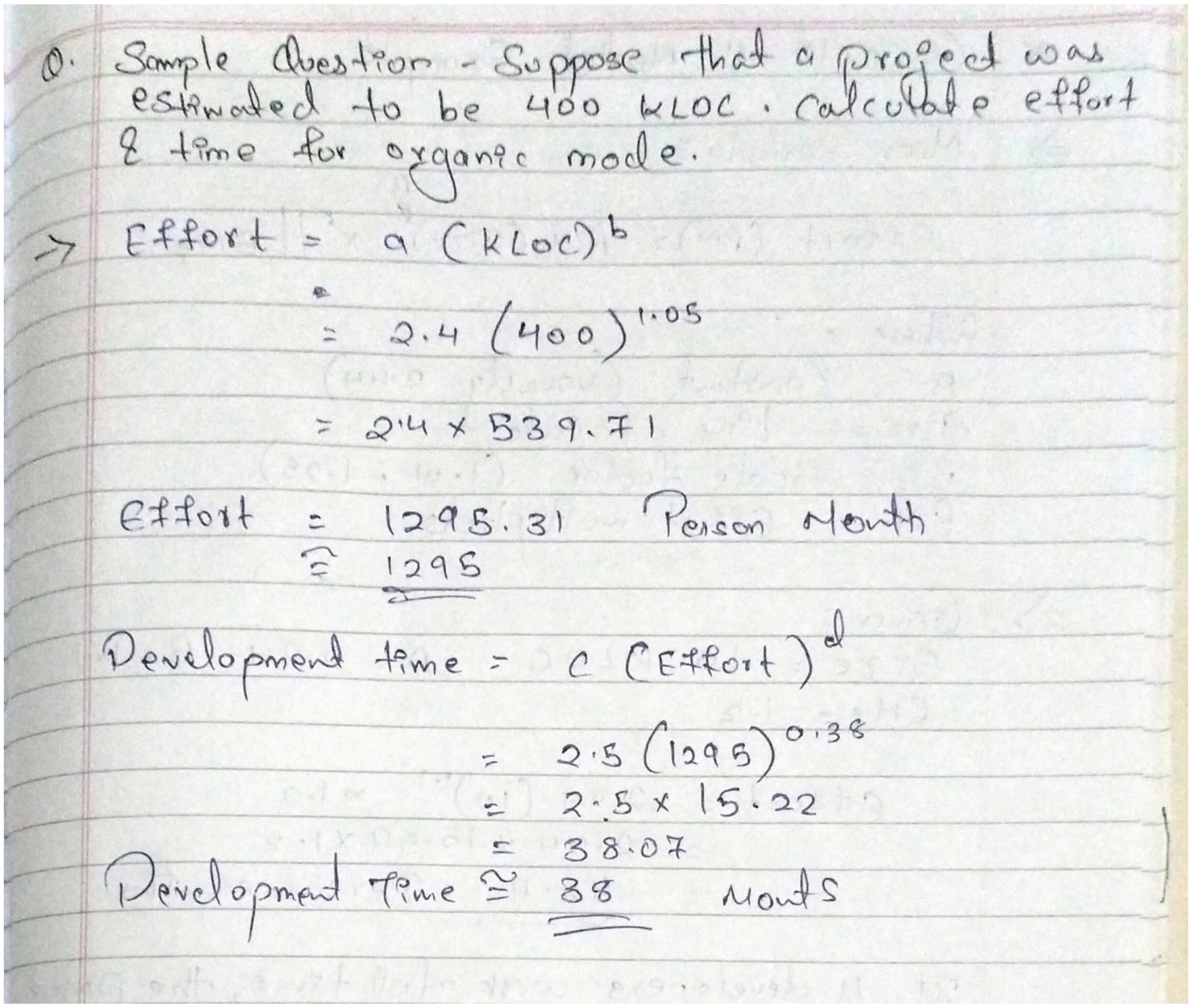
i. Effort = $a\,(KLOC)^b$ Person Month

ii. Development Time = $c\,(Effort)^d$ Months

iii. Average Staff Size = $\frac{Effort}{Dev.\ time}$ Persons

iv. Productivity = $\frac{KLOC}{Effort}$ KLOC / P.M

Mode	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32



Q. Sample Question - Suppose that a project was estimated to be 400 KLOC. Calculate effort & time for organic mode.

→ Effort $= a\,(\text{KLOC})^b$

$= 2.4\,(400)^{1.05}$

$= 2.4 \times 539.71$

Effort $= 1295.31$ Person Month

$\cong \underline{1295}$

Development time $= c\,(\text{Effort})^d$

$= 2.5\,(1295)^{0.38}$

$= 2.5 \times 15.22$

$= 38.07$

Development Time $\cong \underline{38}$ Months

3. Advantages:

- Provides a **quantitative and structured** estimation method.
- Helps in **project scheduling, budgeting, and risk analysis**.
- Adaptable for **different project types**.

4. Limitations:

- Depends heavily on **accurate KLOC estimation**.
- Doesn't consider **modern Agile or component-based development** directly.

- Based on historical data, so accuracy may vary for new technologies.

Q3. [10 Marks] - Answers

a. Explain Function Point Cost Estimation Technique with example.

Function Point Cost Estimation Technique – Explained

Function Point Analysis (FPA) is a **software estimation technique** used to measure **the size of a software system** based on the **functionality** it provides to the **user**, *not* on lines of code.

It helps estimate:

- Project size
- Cost
- Effort
- Time
- Productivity

Why Function Points Instead of LOC?

Unlike Lines of Code (LOC), **Function Points** are:

- Language independent
- Based on user requirements
- Useful early in SDLC (even before coding begins)

Five Major Components of Function Points

Function points are calculated by counting five types of user-visible functions:

1. External Inputs (EI)

Data entering the system (e.g., login form input, add-user form).

2. External Outputs (EO)

Data leaving the system (e.g., reports, statements).

3. External Inquiries (EQ)

Requests that require immediate response (e.g., search feature).

4. Internal Logical Files (ILF)

Files maintained by the system (e.g., employee DB table).

5. External Interface Files (EIF)

Files used but not maintained by the system (e.g., an external bank DB accessed for verification).

Each function is rated **Low, Average, or High** → assigned predefined weights.

* SEN-2024-May

=> Function Point cost Estimation Technique example.

Number of External Inputs (I) =	30
Number of External output (O) =	60
Number of External enquiries (E) =	23
Number of files (F) =	08
Number of External interfaces (N) =	02

constants

$I \times 4 = 30 \times 4 = 120$
$O \times 5 = 60 \times 5 = 300$
$E \times 4 = 23 \times 4 = 92$
$F \times 10 = 8 \times 10 = 80$
$N \times 7 = 2 \times 7 = 14$

=> Calculating UFP (Unadjusted function Point)

Formula : $UFP = \Sigma(\text{Count} \times \text{weight})$

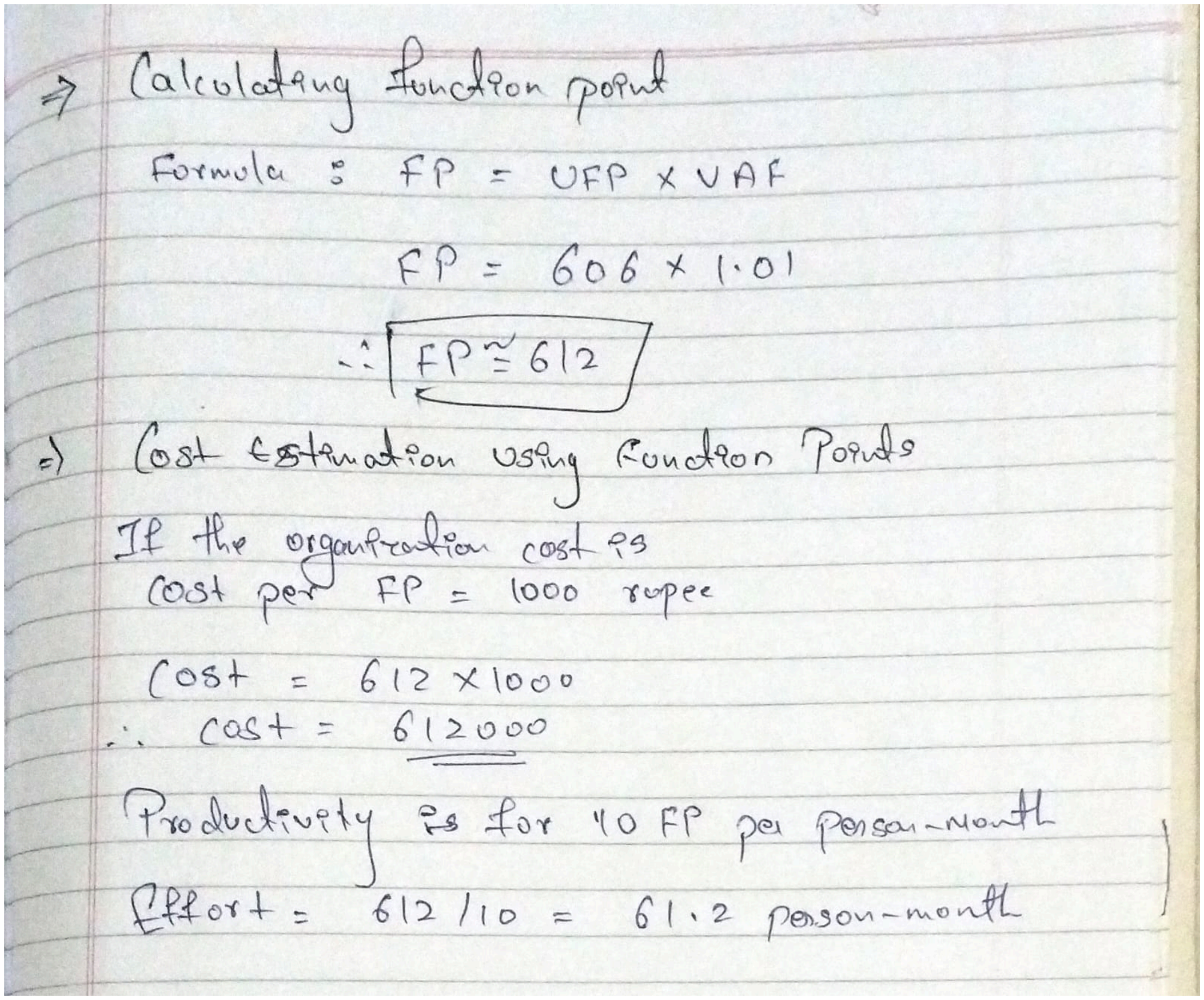
$\therefore \boxed{UFP = 606}$

=> Calculating VAF (Value Adjustment factor)

Formula : $VAF = 0.65 + (0.01 \times \Sigma f_i)$

Given : $\Sigma f_i = 36$

$\therefore VAF = 0.65 + (0.01 \times 36)$
$= 0.65 + (0.36)$
$\boxed{VAF = 1.01}$



⇒ Calculating function point

Formula : $FP = UFP \times VAF$

$FP = 606 \times 1.01$

$\therefore \boxed{FP \cong 612}$

⇒ Cost estimation using function Points

If the organization cost is

Cost per FP = 1000 rupee

$Cost = 612 \times 1000$

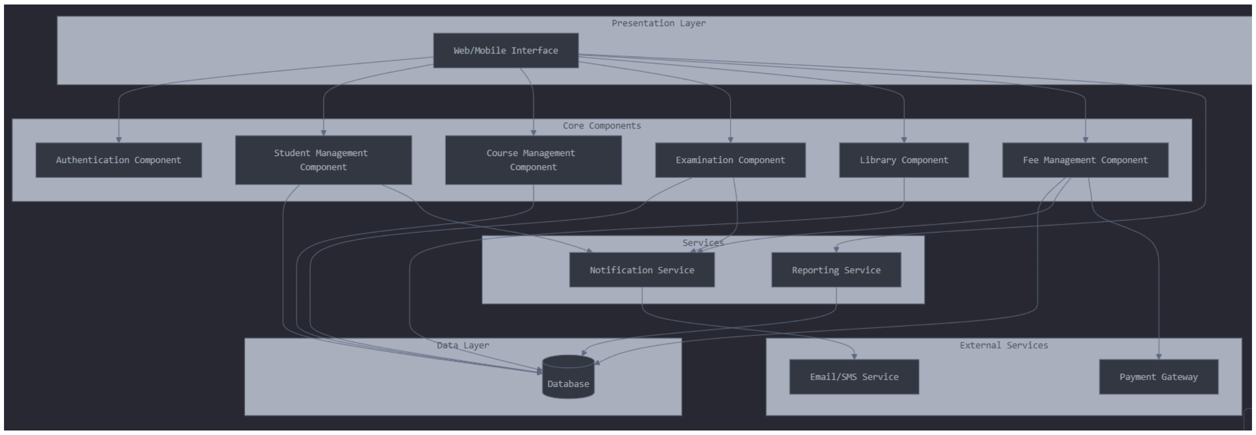
$\therefore$ cost = $\underline{\underline{612000}}$

Productivity is for 10 FP per person-month

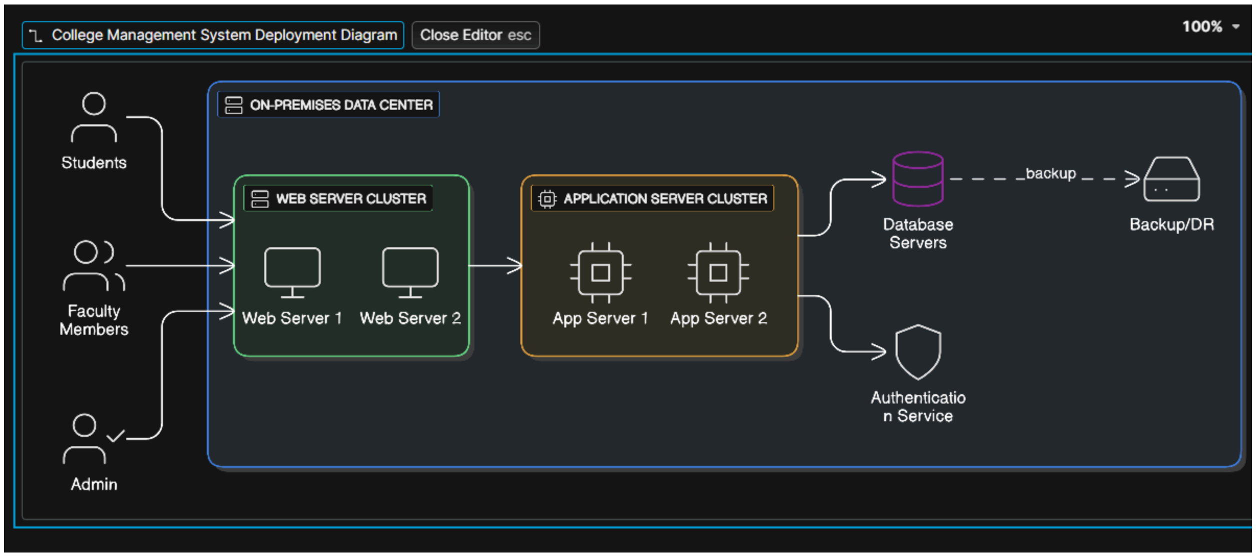
Effort = 612 / 10 = 61.2 person-month

b. Draw UML Component Diagram and Deployment Diagram for "College Management System".

UML Component Diagram:



Deployment Diagram:



Q4. [10 Marks] - Answers

a. Explain Formal Technical Review (FTR) in detail.

Formal Technical Review (FTR)

A Formal Technical Review (FTR) is a structured and peer-driven process used to evaluate the technical quality of software artifacts such as requirements, design, code, or test plans. It helps identify defects early and ensures compliance with standards.

Objectives of FTR

- Detect errors in software work products.
- Ensure adherence to standards and specifications.
- Improve software quality and maintainability.
- Promote team learning and knowledge sharing.

Steps in the FTR Process

1. **Planning**

- Select the work product to be reviewed (e.g., design document).
- Assign roles: moderator, author, reviewers, recorder.
- Schedule the review meeting.

2. **Preparation**

- Reviewers study the document independently.
- Use checklists to identify potential issues.

3. **Conducting the Review**

- Moderator leads the meeting.
- Author presents the work product.
- Reviewers discuss findings and record defects.
- Focus is on technical correctness, not personal criticism.

4. **Rework**

- Author corrects the identified defects.
- Changes are documented and verified.

5. **Follow-Up**

- Moderator ensures all issues are resolved.
- Review status is updated (e.g., accepted, needs re-review).

Roles in FTR

Role	Responsibility
Moderator	Facilitates the review process
Author	Presents and explains the work product
Reviewer	Evaluates the product and finds defects
Recorder	Documents issues and decisions

b. Discuss the various types of design patterns.

What Are Design Patterns?

Design patterns are proven solutions to common software design problems. They provide templates for writing maintainable, scalable, and reusable code. Patterns are not code but **best practices**.

Types of Design Patterns

Design patterns are broadly classified into **three categories**:

1. **Creational Patterns**

Focus on object creation mechanisms, aiming to create objects in a manner suitable to the situation.

- **Singleton**: Ensures a class has only one instance and provides a global access point.
- **Factory Method**: Defines an interface for creating objects but lets subclasses decide which class to instantiate.
- **Abstract Factory**: Provides an interface to create families of related objects without specifying their concrete classes.
- **Builder**: Separates object construction from its representation.
- **Prototype**: Creates new objects by copying an existing object (cloning).

2. **Structural Patterns**

Deal with object composition and relationships to form larger structures.

- **Adapter**: Converts one interface into another expected by the client.
- **Bridge**: Separates abstraction from implementation so both can vary independently.
- **Composite**: Treats individual objects and compositions uniformly.
- **Decorator**: Adds responsibilities to objects dynamically.
- **Facade**: Provides a simplified interface to a complex subsystem.
- **Flyweight**: Reduces memory usage by sharing common data among similar objects.
- **Proxy**: Controls access to another object (e.g., remote, virtual, or protection proxy).

3. **Behavioral Patterns**

Focus on communication between objects and responsibility delegation.

- **Observer**: One-to-many dependency; when one object changes, others are notified.
- **Strategy**: Defines a family of algorithms and makes them interchangeable.
- **Command**: Encapsulates a request as an object.

- **State**: Allows an object to change its behavior when its internal state changes.
- **Template Method**: Defines the skeleton of an algorithm, deferring steps to subclasses.
- **Iterator**: Provides a way to access elements sequentially without exposing the underlying structure.
- **Mediator**: Centralizes complex communication between objects.
- **Chain of Responsibility**: Passes requests along a chain until handled.

Q5. [10 Marks] - Answers

a. Explain in detail the Software Configuration Management process and benefits of SCM.

Software Configuration Management (SCM) – Process

SCM is a discipline that manages changes in software to maintain integrity, traceability, and control throughout the development lifecycle.

Key Steps in the SCM Process

1. **Configuration Identification**

- Define and label configuration items (CIs) like code, documents, test cases.
- Establish naming conventions and versioning rules.

2. **Configuration Control**

- Manage changes through a formal change control process.
- Use Change Control Board (CCB) to evaluate and approve modifications.

3. **Configuration Status Accounting**

- Track and report the status of configuration items.
- Maintain logs of versions, changes, and approvals.

4. **Configuration Auditing**

- Verify that CIs comply with specifications and standards.
- Ensure all changes are properly implemented and documented.

5. **Version Control**

- Maintain multiple versions of software artifacts.
- Use tools like Git, SVN to manage updates and rollback if needed.

6. **Build Management**

- Automate and control the process of compiling and packaging software.
- Ensure consistency across builds and environments.

7. **Release Management**

- Plan and control software releases to users.
- Include release notes, deployment instructions, and rollback plans.

Benefits of SCM

1. **Improved Change Management**

- Tracks every change with proper documentation and approval.

2. **Version Control**

- Prevents conflicts and loss of work by managing multiple versions.

3. **Enhanced Team Collaboration**

- Enables multiple developers to work on shared codebase safely.

4. **Traceability**

- Links changes to requirements, bugs, or enhancements.

5. **Quality Assurance**

- Ensures only verified and approved changes are released.

Q6. [10 Marks] - Answers

a. Explain what is a risk? Different types of risk? and describe RMMM in detail?

What is a Risk in Software Engineering?

A **risk** is a potential problem that may negatively impact the success of a software project. It represents uncertainty that could affect cost, schedule, quality, or performance.

- Risks can arise from technical challenges, resource limitations, changing requirements, or external factors.
- Managing risks early helps prevent project failure or delays.

Types of Risks

1. **Project Risks**

- Affect project schedule, budget, or resources.
- Example: Unrealistic deadlines, team turnover.

2. **Technical Risks**

- Related to technology, tools, or implementation.
- Example: Integration issues, performance bottlenecks.

3. **Business Risks**

- Impact the product’s market success or customer satisfaction.
- Example: Product not meeting user needs, market changes.

4. **Operational Risks**

- Arise from internal processes or infrastructure.
- Example: Server downtime, deployment failures.

RMMM – Risk Mitigation, Monitoring, and Management

RMMM is a structured approach to handle risks throughout the software development lifecycle.

1. **Risk Identification**

- List all possible risks using brainstorming, checklists, or historical data.

2. **Risk Analysis**

- Assess each risk based on:
    - **Probability** (likelihood of occurrence)
    - **Impact** (severity of consequences)

3. **Risk Prioritization**

- Rank risks using a **Risk Exposure** formula: Risk

$$RiskExposure = Probability \times Impact$$

4. **Risk Mitigation**

- Develop strategies to reduce risk probability or impact.

- Example: Use proven technologies, allocate buffer time.

5. **Risk Monitoring**

- Track risk status regularly.
- Use metrics, reviews, and reports to detect changes.

6. **Risk Management**

- Take corrective actions if risks materialize.
- Update plans and communicate with stakeholders.

b. Why is cyclomatic complexity important to testers?

Importance of Cyclomatic Complexity to Testers

Definition:

Cyclomatic Complexity (CC) is a metric that measures the **number of independent paths** in a program’s source code.

It helps determine the **complexity** and **testability** of the software.

Importance to Testers:

1. **Determines Minimum Number of Test Cases:**
    - CC gives the **minimum number of test cases** needed for **complete branch coverage** (testing all possible paths).

2. **Identifies Complex Modules:**
    - High CC indicates complex or error-prone code.
    - Testers can **focus more testing effort** on these areas.

3. **Improves Test Coverage:**
    - Ensures that **every decision point (if, while, for, switch)** is tested at least once.

4. **Aids in Maintenance and Risk Analysis:**
    - Modules with high complexity are harder to maintain and more likely to contain defects.
    - Testers can mark them as **high-risk areas**.

1. How many predicate nodes are there and what are their names?

Predicate nodes are nodes in the flow graph where control can branch in two or more directions; these correspond to decision points in a program.

Answer:

There are 3 predicate nodes:

- Node 1
- Node 6
- Node (2,3) (based on the visual, it's a combined decision node labeled (2,3)).

2. How many regions are there in flow graph F?

Regions in a flow graph are the areas enclosed by edges, including the area outside.

Answer:

There are 4 regions in flow graph F.

3. What is the cyclomatic complexity of flow graph F?

Cyclomatic complexity (V(G)/) can be found using the formula:

$$V(G) = E - N + 2$$

where E = number of edges, N = number of nodes. It also equals the number of regions or number of predicate nodes plus one.

Answer:

The cyclomatic complexity of flow graph F is 4.

4. How many nodes are there in the longest independent path?

The longest independent path is the path from entry to exit passing through the maximum number of nodes without repetition.

Answer:

There are 8 nodes in the longest independent path from node 1 to node 11.

5. How many nodes are there in flow graph F?

Count each node in the flow graph diagram.

Answer:

There are 9 nodes in flow graph F.