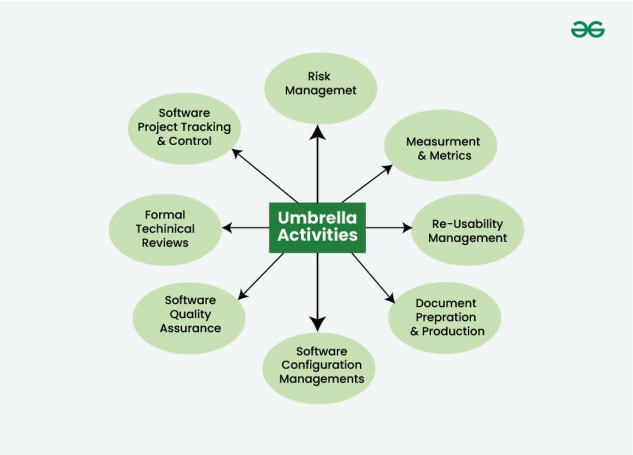
Q. 1 Solve any Four out of the following (5 marks each) 20M

a. Explain Software Process Umbrella Activities

Software engineering is a collection of co-related steps.

These steps are presented or accessed in different approaches in different software process models.
Umbrella activities are a set of steps or procedures that the software engineering team follows to maintain the progress, quality, change and risks of the overall development tasks.
These steps of umbrella activities will evolve through the phases of generic view of software development.



Software project tracking and control: This activity allows the software team to check the progress of software development. Before the actual development starts, make a software development plan and develop on this basis, but after a certain period of time, it is necessary to analyse the development progress to find out what measures need to be taken. It must be accepted at an appropriate time after the completion of development, testing, etc. The test results may need to reschedule the development time.
Formal Technical reviews: Software engineering is done in clusters or modules, after completing each module, it is good practice to review the completed module to find out and remove errors so their propagation to the next module can be prevented.
Software quality assurance: The quality of the software, such as user experience, performance, workload flexibility, etc., must be tested and verified after reaching the specified milestones, which reduces the tasks at the end of the development process, which must be performed by a dedicated team so that the development can continue.
Software configuration management: It manages the impact of changes throughout the software development process. Software Configuration Management (SCM) is a set of activities designed to manage changes by identifying work products that can be changed, establishing relationships between them, and defining mechanisms for managing different versions of them. Work product.
Document preparation and production: All the project planning and other activities should be hard copied and the production get started here.

Reusability management: This includes the approval of any part of a backing-up software project or any type of support provided for updates or updates in the future. Update the software according to user/current time requirements.
Measurement: This includes all measurements of all aspects of the software project.
Direct measures: cost,line of code, size of team and software
Indirect measures: quality of software (efficiency, usability etc)
Risk management: Risk management is a series of steps to help software development teams understand and manage uncertainty. It is a very good idea to identify it, assess the likelihood of it happening, assess its impact, and develop an "if the problem does happen" contingency plan.

b. Explain software reengineering

Software Re-Engineering is the examination and alteration of a system to reconstitute it in a new form. The principle of Re-Engineering when applied to the software development process is called software re-engineering. It positively affects software cost, quality, customer service, and delivery speed. In Software Re-engineering, we are improving the software to make it more efficient and effective.

It is a process where the software's design is changed and the source code is created from scratch. Sometimes software engineers notice that certain software product components need more upkeep than other components, necessitating their re-engineering.

The re-Engineering procedure requires the following steps

1. Decide which components of the software we want to re-engineer. Is it the complete software or just some components of the software?
2. Do Reverse Engineering to learn about existing software functionalities.
3. Perform restructuring of source code if needed for example modifying functional-Oriented programs in Object-Oriented programs
4. Perform restructuring of data if required
5. Use Forward Engineering ideas to generate re-engineered software

Software Re-Engineering Activities:

1. Inventory Analysis:

Every software organization should have an inventory of all the applications. Inventory can be nothing more than a spreadsheet model containing information that provides a detailed description of every active application.

2. Document reconstructing:

Documentation of a system either explains how it operates or how to use it. Documentation must be updated. It may not be necessary to fully document an application. The system is business-critical and must be fully re-documented.

3. Reverse Engineering:

Reverse engineering is a process of design recovery. Reverse engineering tools extract data and architectural and procedural design information from an existing program.

4. Code Reconstructing:

To accomplish code reconstruction, the source code is analyzed using a reconstructing tool. Violations of structured programming construct are noted and code is then reconstructed. The resultant restructured code is reviewed and tested to ensure that no anomalies have been introduced.

5. Data Restructuring:

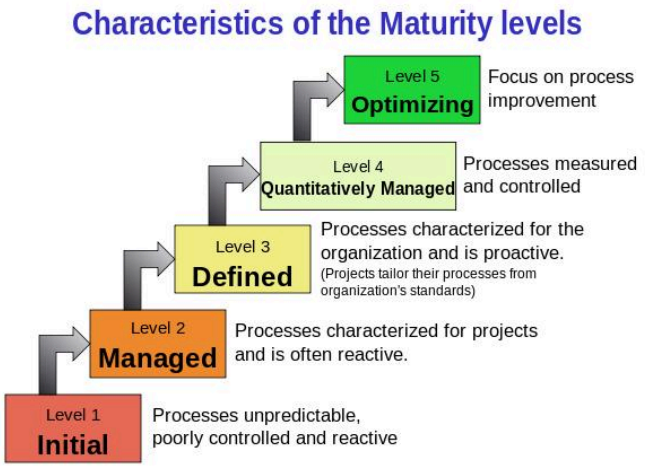
Data restructuring begins with a reverse engineering activity. The current data architecture is dissected, and the necessary data models are defined. Data objects and attributes are identified, and existing data structures are reviewed for quality.

6. Forward Engineering:

Forward Engineering also called renovation or reclamation not only recovers design information from existing software but uses this information to alter or reconstitute the existing system to improve its overall quality.

c. What is Capability Maturity Model (CMM) Explain different CMM levels

The Capability Maturity Model (CMM) is a tool used to improve and refine software development processes. It provides a structured way for organisations to assess their current practices and identify areas for improvement. CMM consists of five maturity levels: initial, repeatable, defined, managed, and optimising. By following the CMM, organisations can systematically improve their software development processes, leading to higher-quality products and more efficient project management.



Maturity Level 1 – Initial: Work Performed informally.
Company has no standard process for software development. processes are disorganized, ad hoc and even chaotic.
Nor does it have a project-tracking system that enables developers to predict costs or finish dates with any accuracy. startup.
Maturity Level 2 – Managed/ Repeatable: Work is planned and Tracked
Company has installed basic software management processes and controls to track cost, schedule, and functionality..
The process is in place to repeat the earlier successes on projects with similar applications.
Maturity Level 3 – Defined: Work is well defined
Company has pulled together a standard set of processes and controls for the entire organization so that developers can move between projects more easily and customers can begin to get consistency from different groups.
Maturity Level 4–Quantitatively Managed: Work id Quantitatively Controlled
In addition to implementing standard processes, the company has installed systems to measure the quality of those processes across all projects.
Maturity Level 5 – Optimizing: Continuous Improvement
Company has accomplished all of the above and can now begin to see patterns in performance over time, so it can tweak its processes in order to improve productivity and reduce defects in software development across the entire organization.

d. Design User Interface for Online Shopping System

e. Discuss limitations of Waterfall model & Spiral Model

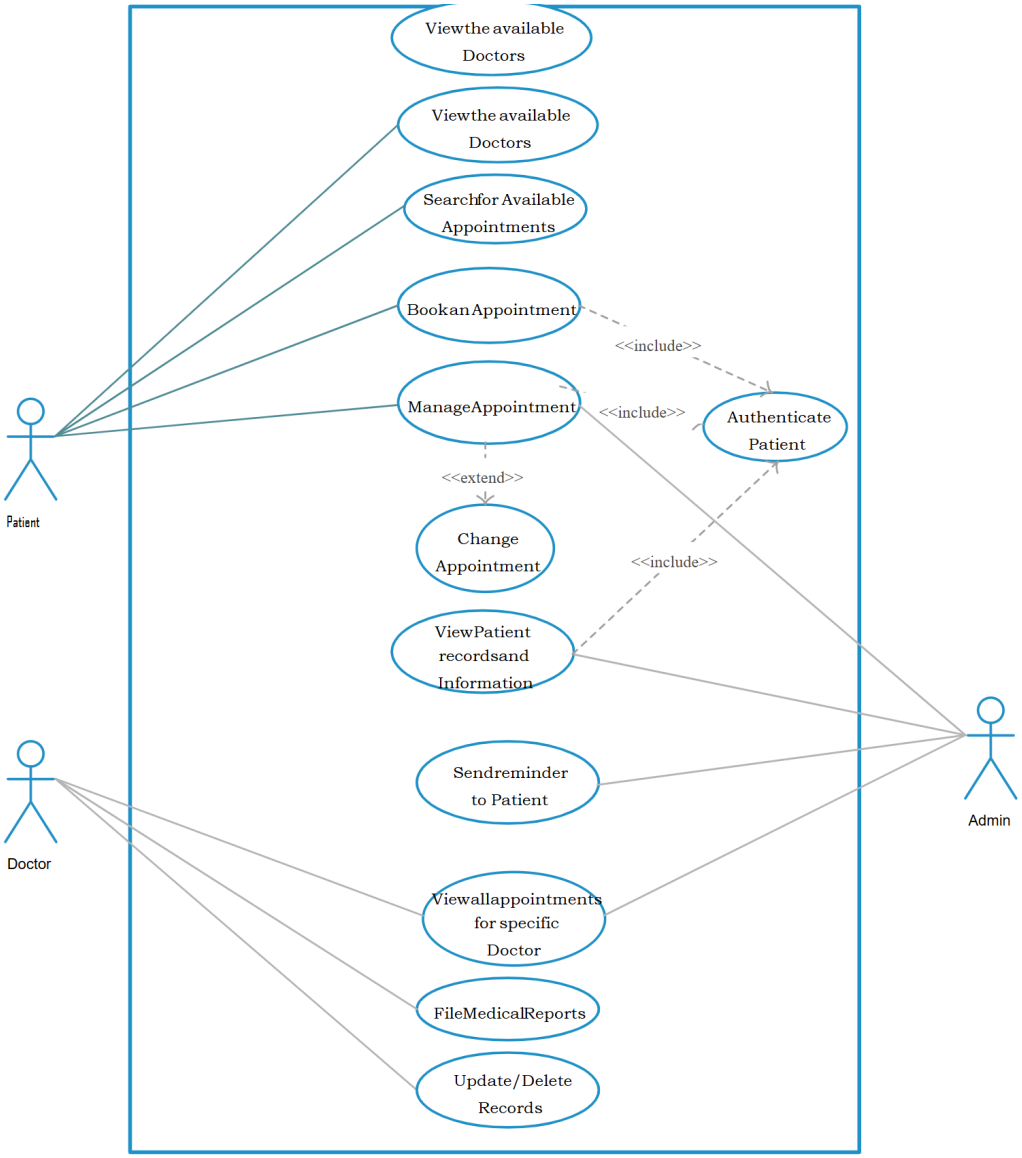
Waterfall Model:

1. “Blocking states” members of one team has to wait for other team members to complete the dependant tasks(development=>modelling)
2. No working software is produced until late during the life cycle.
3. High amounts of risk and uncertainty.
4. Not a good model for complex projects.
5. Poor model for long and ongoing projects.
6. Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
7. It is difficult to measure progress within stages.
8. Cannot accommodate changing requirements.

Spiral Model:

1. If major risk is not discovered in early Iteration of spiral, it may become a major risk in later stages
2. It is not suitable for the small and low-risk product because it could be costly for a smaller project.
3. Rules and protocols must be followed very strictly to implement the approach
4. In the spiral model, management is a bit difficult; that's why it is a complex process.
5. The maximum number of intermediate phases needs unnecessary paperwork. [Time Consuming].

f. Draw Use Case Diagram for Hospital Management System



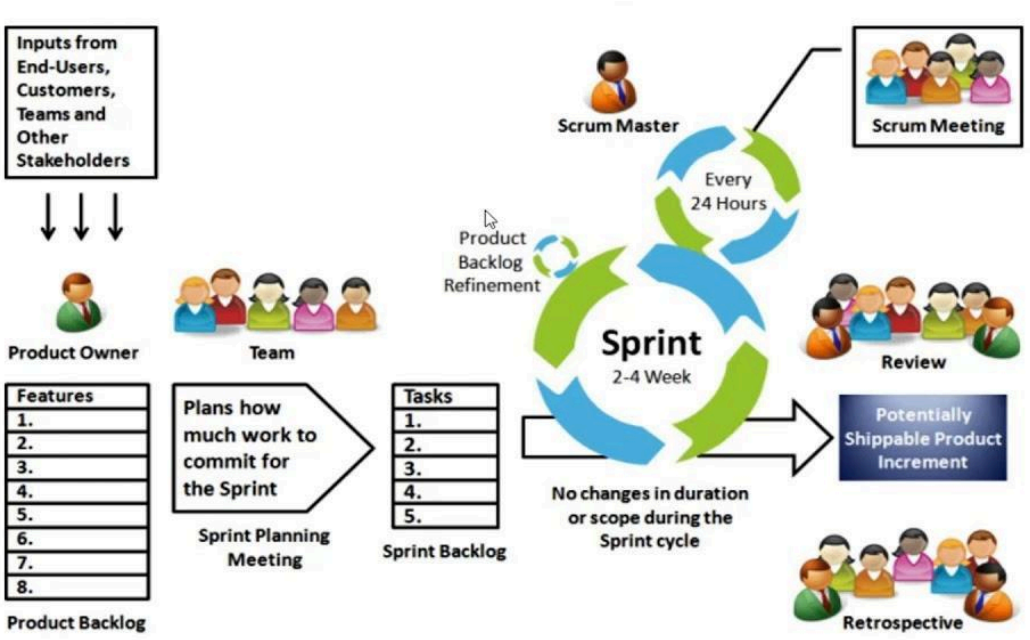
Q. 2

a. What is Agile Process? Explain SCRUM Process Model with all activities 10M

In software Development, **Agile** means ability to respond quickly to changes – changes in Requirements, Technology or people.
Agile SDLC model is a combination of iterative and incremental process model with focus on customer satisfaction by rapid delivery of working software products.
Direct collaboration with customers.
Agile methods break the product into smaller incremental builds, these builds are provided in increments.
Each iteration typically lasts from about 1 to 3 weeks.
Agile model believes that every project needs to be handled differently and existing methods need to be tailored to best suit the project requirements
In agile, tasks are divided into time boxes to deliver specific features for the release

time box: a defined period of time during which a task must be accomplished
Iterative approach is taken and working software build is delivered after each iteration
Each build is incremental in terms of features, the final build holds all the features required by the customer.

Scrum is a subset of Agile.
It is a lightweight process framework for agile development, and the most widely-used one.
Scrum is an innovative approach to getting work done in an efficient way.
It is an Iterative and Incremental Software Development framework
It is agile process model used to develop complex and sustainable products
It enables teams to self organise by collaborations of team members.
Scrum is a framework utilising an agile mindset for developing, delivering, and sustaining products in a complex environment



Different roles are used in the scrum process. Three different roles are as follows:
Product owner: A Product Owner orders the work for a complex problem into a Product Backlog. The Product Owner is responsible for continuously communicating the vision and priorities to the development team.
Development team: The Development Team turns a selection of the work into an Increment of value during a Sprint. The scrum development team is generally size of 5-9 peoples with self-organising and cross-functional skills
Scrum Master: Person responsible for scrum process and helps the team remain creative and productive while making sure its successes are visible to the Product Owner.
Sprints are a short, time-boxed period for the Scrum team that works to complete a set /amount of work. Sprints are the core component of Scrum and agile methodology.
Sprint Planning: At the start of each sprint, a sprint planning meeting is held

All stakeholders plan how much work from the product backlog to be committed for the sprint as per the owner's suggestion.
That work is then moved from the product backlog to a **sprint backlog**
Then work starts to develop. During this period we can get **daily scrum**
All team members are required to attend the daily scrum.
Sprint review meeting: At the end of each sprint, the team demonstrates the completed functionality at a sprint review meeting, during which, the team shows what they accomplished during the sprint.

b. What do you mean by Cohesion & Coupling? Explain different types of cohesion & Coupling 10M

Cohesion:

1. The measure of how strongly the elements are related functionally inside a module is called cohesion
2. A good software design will have high cohesion.
3. In software engineering the elements inside a module can be instructions, groups of instructions, definition of data, call from another module etc.
4. The aim is always for functions that are strongly related and the expectation is for everything inside the module to be in connection with one another.
5. Cohesion is a measure of functional strength of a module.
6. Basically, cohesion is the internal glue that keeps the module together.
7. Good system design must have high cohesion between the components of the system.

TYPES

Coincidental cohesion: (worst): An unplanned cohesion where elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion Eg.- print next line and reverse the characters of a string in a single component.

Logical cohesion: When logically classified elements are combined into a single module but not functionally then it is called logical cohesion. Here all elements of modules contribute to the same system operation. Eg.- Print Functions: the case where a set of print functions generating different output reports are arranged into a single module.

Temporal Cohesion: The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time-span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time. Eg.- a function which is called after

catching an exception which closes open files, creates an error log, and notifies the user.

Procedural Cohesion: A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which a certain sequence of steps have to be carried out for achieving an objective. Eg.- a function which checks file permissions then opens the file.

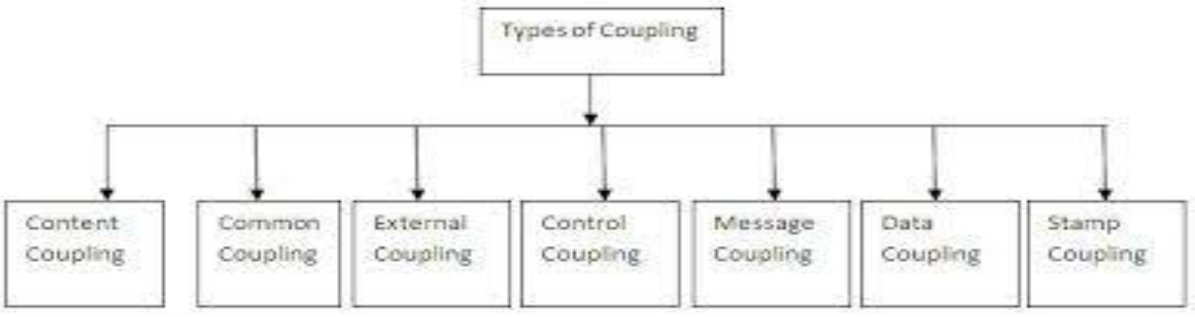
Sequential Cohesion: Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part like an assembly line (e.g. a function which reads data from a file and processes the data). Sequential cohesion is easy maintenance Eg.- In a transaction processing system (TPS), the get-input, validate-input, sort-input functions are grouped into one module.

Communication cohesion: When elements of a module perform different functions but each function accepts the same input and generates the same output then that module is said to be communicational cohesive. Eg.- Module determines customer details like use customer account no to find and return customer name and loan balance.

Functional cohesion (best): If elements of a module are grouped together since they contribute to a single function then the module is functionally cohesive. All elements of such a module are necessary for successful execution of function. Eg.- Read transaction record or Assign seat to airline passenger.

Coupling

1. Coupling is the measure of the degree of interdependence between the modules.
2. A good software will have low coupling.
3. The lower the coupling, the more modular a program is, which means that less code has to be changed when the program's functionality is altered later on.
4. However, coupling cannot be completely eliminated; it can only be minimised.



Content Coupling (Worst): In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. E.g.- a branch from one module into another module.

Common Coupling: The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. Eg.- when two classes access the same shared data (e.g., a global variable).

Control coupling: When one function controls the flow of another function.

Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data.

Stamp Coupling : In stamp coupling, the complete data structure is passed from one module to another module.

External Coupling: In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware.

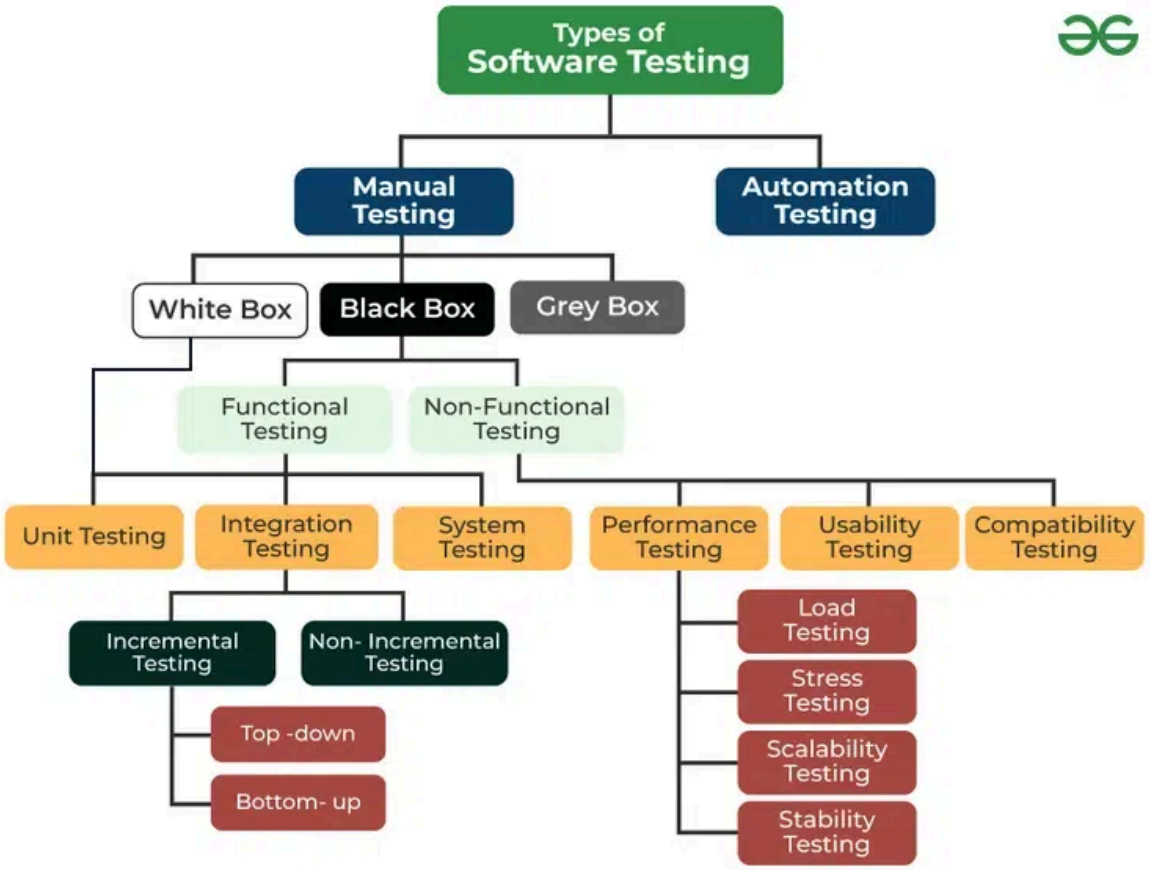
Message Coupling: This type of coupling can be achieved by state decentralisation. It is the loosest type of coupling, in which the component communication is performed through message passing.

Q. 3

a. What is Software Testing? Explain different types of software testing 10M

Software testing is an important process in the software development lifecycle . It involves verifying and validating that a software application is free of bugs, meets the technical requirements set by its design and development , and satisfies user requirements efficiently and effectively.

This process ensures that the application can handle all exceptional and boundary cases, providing a robust and reliable user experience. By systematically identifying and fixing issues, software testing helps deliver high-quality software that performs as expected in various scenarios.



Manual testing is a technique to test the software that is carried out using the functions and features of an application. In manual software testing, a tester carries out tests on the software by following a set of predefined test cases. In this testing, testers make test cases for the codes, test the software, and give the final report about that software. Manual testing is time-consuming because it is done by humans, and there is a chance of human errors.

Automated Testing is a technique where the Tester writes scripts on their own and uses suitable Software or Automation Tool to test the software. It is an Automation Process of a Manual Process. It allows for executing repetitive tasks without the intervention of a Manual Tester.

White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

Black-box testing is a type of software testing in which the tester is not concerned with the internal knowledge or implementation details of the software but rather focuses on validating the functionality based on the provided specifications or requirements.

Gray Box Testing is a software testing technique that is a combination of the Black Box Testing technique and the White Box Testing technique. The internal structure is partially known.

Unit testing involves the testing of each unit or an individual component of the software application. It is the first level of functional testing. The aim behind unit testing is to validate unit components with its performance. A unit is a single testable part of a software system and tested during the development phase of the application software. The purpose of unit testing is to test the correctness of isolated code. A unit component is an individual function or code of the application. White box testing approach used for unit testing and usually done by the developers. Whenever the application is ready and given to the Test engineer, he/she will start checking every component of the module or module of the application independently or one by one, and this process is known as Unit testing or components testing.

Integration testing is the process of testing the interface between two software units or modules. It's focused on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

System Testing (ST) is a black box testing technique performed to evaluate the complete system's compliance against specified requirements. In System testing, the functionalities of the system are tested from an end-to-end perspective. System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased.

Performance Testing is a software testing process used for testing the speed, response time, stability, reliability, scalability and resource usage of a software application under particular workload. The main purpose of performance testing is to identify and eliminate the performance bottlenecks in the software application. It is a subset of performance engineering and also known as “Perf Testing”.

User Acceptance Testing (UAT) is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done. The main Purpose of UAT is to validate end to end business flow. It does not focus on cosmetic errors, spelling mistakes or system testing. User Acceptance Testing is carried out in a separate testing environment with production-like data setup. It is a kind of black box testing where two or more end-users will be involved.

b. Define Risk? What are different categories of risks? Explain RMMM plan 10M with suitable example.

Risk the probability of suffering a loss.
A risk is a potential problem – it might happen and it might not
In project development, the loss illustrates the impact to the project which could be in the form of diminished quality of the end product, increased costs, delayed completion time or failure.

Categories of risk:

SCHEDULE RELATED RISK: refers to time related risks or project delivery related planning risks. The wrong schedule affects the project development and delivery.

BUDGET RISK: refers to the monetary risks mainly it occurs due to budget overruns. Always the financial aspect for the project should be managed as per decided but if financial aspect of project mismanaged then there budget concerns will arise by giving rise to budget risks.

OPERATIONAL RISKS: refers to the procedural risks means these are the risks which happen in day-to-day operational activities during project development due to improper process implementation or some external operational risks.

TECHNICAL RISKS: refers to the functional risk or performance risk which means this technical risk mainly associated with functionality of product or performance part of the software product.

PROGRAMMATIC RISKS: refers to the external risk or other unavoidable risks. These are the external risks which are unavoidable in nature.
These risks come from outside and it is out of control of programs.

A risk management technique is usually seen in the software Project plan. This can be divided into Risk Mitigation, Monitoring, and Management Plan (RMMM). In this plan, all work is done as part of risk analysis. As part of the overall project plan, the project manager generally uses this RMMM plan.
In some software teams, risk is documented with the help of a Risk Information Sheet (RIS). This RIS is controlled by using a database system for easier management of information i.e creation, priority ordering, searching, and other analysis. After documentation of RMMM and start of a project, risk mitigation and monitoring steps will start.

Risk Mitigation :
It is an activity used to avoid problems (Risk Avoidance).
Steps for mitigating the risks as follows.

1. Finding out the risk.
2. Removing causes that are the reason for risk creation.
3. Controlling the corresponding documents from time to time.
4. Conducting timely reviews to speed up the work.

Risk Monitoring :
It is an activity used for project tracking.
It has the following primary objectives as follows.

1. To check if predicted risks occur or not.
2. To ensure proper application of risk aversion steps defined for risk.
3. To collect data for future risk analysis.
4. To allocate what problems are caused by which risks throughout the project.

Risk Management and planning :
It assumes that the mitigation activity failed and the risk is a reality. This task is done by the Project manager when risk becomes reality and causes severe problems. If the project manager effectively uses project mitigation to remove risks successfully then it is easier to manage the risks. This shows the response that will be taken for each risk by a manager. The main objective of the risk management plan is the risk register. This risk register describes and focuses on the predicted threats to a software project.

Example:
Let us understand RMMM with the help of an example of high staff turnover.
Risk Mitigation:
To mitigate this risk, project management must develop a strategy for reducing turnover. The possible steps to be taken are:

- Meet the current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes that are under our control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.
- Organise project teams so that information about each development activity is widely dispersed.
- Define documentation standards and establish mechanisms to ensure that documents are developed in a timely manner.
- Assign a backup staff member for every critical technologist.

Risk Monitoring:
As the project proceeds, risk monitoring activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of high staff turnover, the following factors can be monitored:

- General attitude of team members based on project pressures.
- Interpersonal relationships among team members.
- Potential problems with compensation and benefits.
- The availability of jobs within the company and outside it.

Risk Management:
Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality. Continuing the example, the project is well underway, and a number of people announce that they will be leaving. If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team. In addition, the project manager may temporarily refocus resources (and readjust the project schedule) to those functions that are fully staffed, enabling newcomers who must be added to the team to “get up to the speed“

Q. 4

a. Explain & compare FTR & Walkthrough. 10M

Formal Technical Review (FTR) is an activity performed by software engineers to give assurance of software quality.

Objectives of formal technical review (FTR):

1. Useful to uncover error in logic, function and implementation for any representation of the software.
2. The purpose of FTR is to verify that the software meets specified requirements.
3. To ensure that software is
4. according to predefined standards.
5. It helps to review the uniformity in software that is developed in a uniform manner.
6. To make the project more manageable.

Steps in FTR:-

The review meeting: Every review meeting should be conducted by considering the following constraints-

- Involvement of people
- Advance preparation should occur but it should be very short that is at the most 2 hours of work for each person
- short duration of the review meeting should be less than two hours.

Review reporting and record keeping:

- During the FTR, the reviewer actively records all the issues that have been raised.
- At the end of the meeting these all raised issues are consolidated and a review issue list is prepared.

Review guidelines:

• Guidelines for the conducting of formal technical review must be established in advance. These guidelines must be distributed to all reviewers, agreed upon, and then followed.

Walkthrough: Method of conducting informal group/individual review is called walkthrough, in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems or may suggest improvement on the article, walkthrough can be pre planned or can be conducted at need basis and generally people working on the work product are involved in the walkthrough process.

The Purpose of walkthrough is to:

1. Find problems
2. Discuss alternative solutions
3. Focusing on demonstrating how work product meets all requirements.IEEE 1028 recommends three specialist roles in a walkthrough:

Leader: who conducts the walkthrough, handles administrative tasks, and ensures orderly conduct (and who is often the Author).

Recorder: who notes all anomalies (potential defects), decisions, and action items identified during the walkthrough meeting, normally generate minutes of meeting at the end of walkthrough session.

Author: who presents the software product in a step-by-step manner at the walk-through meeting, and is probably responsible for completing most action items.

b. Explain change control & Version Control 10M

SCM is also known as software control management. SCM aims to control changes introduced to large complex software systems through reliable version selection and version control.

Version Control:-

Software Version Control is a system or tool that captures the changes to a source code element: files, folders, images or binaries.
A version control system (also known as a Revision Control System) is a repository of files, often the files for the source code of computer programs, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixed, or enhancements introduced, by the change.
Version control systems are essential for any form of distributed, collaborative development. Whether it is the history of a wiki page or large software development project, the ability to track each change as it was made, and to reverse changes when necessary can make all the difference between a well managed and controlled process and an uncontrolled ‘first come, first served’ system. It can also serve as a mechanism for due diligence for software projects. Combines procedures and tools to manage the different versions of configuration objects created during the software process.

• Version control systems require the following capabilities.

- Project repository – stores all relevant configuration objects.
- Version management capability – stores all versions of a configuration object (enables any version to be built from past versions)
- Make facility – enables collection of all relevant configuration objects and constructs a specific software version.
- Issues (bug) tracking capability – enables teams to record and track status of outstanding issues for each configuration object.

Change Control:-

Change control is a systematic approach to managing all changes made to a product or system. The purpose is to ensure that no unnecessary changes are made, that all changes are documented, that services are not unnecessarily disrupted and that resources are used efficiently.

Here's an example of a six-step process for a software change request:

Documenting the change request: When the client requests the change, that request is categorised and recorded, along with informal assessments of the importance of that change and the difficulty of implementing it.

Formal assessment: The justification for the change and risks and benefits of making/not making the change are evaluated. If the change request is accepted, a development team will be assigned. If the change request is rejected, that fact is documented and communicated to the client.

Planning: The team responsible for the change creates a detailed plan for its design and implementation, as well as a plan for rolling back the change should it be deemed unsuccessful.

Designing and testing: The team designs the program for the software change and tests it. If the change is deemed successful, the team requests approval and a date for implementation.

Implementation and review: The team implements the program and stakeholders review the change.

Final assessment: If the client is satisfied that the change was implemented satisfactorily, the change request is closed. If the client is not satisfied, the project is reassessed and steps may be repeated.

Q.5

a. Explain different types of software maintenance. 10M

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance task as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

Corrective Maintenance: Corrective maintenance deals with the repair of faults or defects found in day-today system functions. A defect can result due to errors in software design, logic and coding. Design errors occur when changes made to the software are incorrect, incomplete, wrongly communicated, or the change request is misunderstood. Logical errors result from invalid tests and conclusions, incorrect implementation of design specifications, faulty logic flow, or incomplete test of data. Corrective maintenance accounts for 20% of all the maintenance activities.

Adaptive Maintenance: Adaptive maintenance is the implementation of changes in a part of the system, which has been affected by a change that occurred in some other part of the system. Adaptive maintenance consists of adapting software to changes in the environment such as the hardware or the operating system. Adaptive maintenance accounts for 25% of all the maintenance activities.

Perfective Maintenance: Perfective maintenance mainly deals with implementing new or changed user requirements. Perfective maintenance involves making functional enhancements to the system in addition to the activities to increase the system's performance even when the changes have not been suggested by faults.This includes enhancing both the function and efficiency of the code and changing the functionalities of the system as per the users' changing needs. Perfective maintenance accounts for 50%, that is, the largest of all the maintenance activities.

Preventive Maintenance: Preventive maintenance involves performing activities to prevent the occurrence of errors. It tends to reduce the software complexity thereby improving program understandability and increasing software maintainability. It comprises documentation updating, code optimization, and code restructuring. Preventive maintenance is limited to the maintenance organization only and no external requests are acquired for this type of maintenance. Preventive maintenance accounts for only 5% of all the maintenance activities.

b. What is SRS? Prepare a SRS for Online Movie Booking System. 10M

SRS: A document that outlines the requirements, expectations, design, and standards for a software project. It's a blueprint for the software that describes what it will do and how it will perform. An SRS is a key reference for the development team and other stakeholders. It helps to ensure that the software meets the needs of the business and users, and that it's developed efficiently and cost-effectively.

Software Requirements Specification (SRS) for Online Movie Booking System

1. Introduction

1.1 Purpose
The purpose of this document is to outline the requirements for an Online Movie Booking System, designed to facilitate the booking of movie tickets online. The system will enable users to browse available movies, check showtimes, select seats, and book tickets, offering convenience and efficiency for users.

1.2 Scope
This Online Movie Booking System will allow users to:
Browse movies and view details (showtimes, ratings, trailers, etc.).
Select preferred seats and showtimes.
Make payments securely.
Receive e-tickets and booking confirmation via email.
The system is aimed at cinema operators and customers, aiming to reduce wait times, minimise errors, and increase user satisfaction.

1.3 Definitions, Acronyms, and Abbreviations
User: Anyone who uses the Online Movie Booking System.
Admin: Administrator responsible for managing movies, schedules, and other system functionalities.
SRS: Software Requirements Specification.
UI: User Interface.
1.4 Overview
This document outlines the system functionality, user requirements, and technical requirements for the development of an Online Movie Booking System.

2. Overall Description

2.1 Product Perspective
The Online Movie Booking System is an independent, web-based platform that connects users with movie theaters for seamless ticket bookings.

2.2 Product Functions
User Management: Registration, login, profile management.
Movie and Show Management: Display movies, schedules, and related information.
Seat Selection: View available seats and choose preferences.
Payment Gateway: Secure transactions with various payment options.

2.3 User Classes and Characteristics
Registered Users: Users who can book tickets and manage profiles.
Guests: Users who can view information but need to register to book tickets.
Admin: Manages the system by adding movies, managing schedules, and overseeing bookings.

2.4 Operating Environment
Web application compatible with modern browsers (Chrome, Firefox, Safari, etc.).
Mobile-responsive design for smartphone and tablet users.

2.5 Design and Implementation Constraints
Compliance with industry-standard security protocols.
Integration with third-party payment gateways.

3. Functional Requirements

3.1 User Registration and Login
Users can register by providing personal details and login credentials.
A login function with password recovery options.

3.2 Movie Listings and Search
The system displays a list of currently available movies.
Users can search for movies based on title, genre, or rating.
Each movie page shows details such as trailers, descriptions, reviews, and showtimes.

3.3 Showtimes and Seat Selection
Users can view showtimes for selected movies and select preferred showtimes.
Users can view seating layouts and choose available seats.

3.4 Ticket Booking and Confirmation
Users can confirm the booking by reviewing selected seats, date, and time.
System sends confirmation emails with e-tickets and booking details after successful payment.

3.5 Payment Processing
Secure payment integration (credit card, debit card, PayPal, etc.).
The system will notify users of successful or failed transactions.

4. Non-Functional Requirements

4.1 Security
Encrypted user data and secure payment gateways.
Role-based access for admins and regular users.

4.2 Usability
User-friendly and intuitive interface.
Mobile responsiveness for compatibility with various devices.

4.3 Performance
Fast loading times for pages and database queries.
Handle up to 500 concurrent users without performance degradation.

4.4 Reliability
System uptime of 99.9% or higher.
Backup and recovery procedures to prevent data loss.

4.5 Maintainability
Modular design to allow for easy updates and maintenance.
Well-documented code to facilitate future changes.

Q. 6

a. List different metrics used for software measurement? Explain function 10M point-based estimation technique in detail.

Software metrics can be classified into three categories:

1. Product metrics are software product measures at any stage of their development, from requirements to established systems. Product metrics are related to software features only. Product metrics fall into two classes:
– Dynamic metrics that are collected by measurements made from a program in execution.
– Static metrics that are collected by measurements made from system representations such as design or documentation.

2. Process Metrics: These are the metrics pertaining to the Process Quality.
They measure efficiency and effectiveness of various processes.
Private process metrics (e.g. defect rates by individual or module) are known only to the individual or team concerned.
Public process metrics enable organizations to make strategic changes to improve the software process.
Metrics should not be used to evaluate the performance of individuals.

3. Project metrics: Software project metrics are used by the software team to adapt project workflow and technical activities.
Project metrics are used to avoid development schedule delays, to mitigate potential risks, and to assess product quality on an on-going basis.
Every project should measure its inputs (resources), outputs (deliverables), and results (effectiveness of deliverables).

Function Point Metrics are also known as Function-Oriented Metrics.
This method is actually independent of the programming language that is being used in software applications and based on calculating the Function Point (FP).
This model generally focuses on the functionality of the software application being delivered.
A function point is a unit of measurement that measures the business functionality provided by the business product.
It measures functionality from user's point of view
FPs of an application is found out by counting the number and types of functions used in the applications.

1. Data Functions – Internal Logic or External Interface File
2. Transaction Functions – External Input, External Output, or External Inquiry

Various functions used in an application can be put under five types,

1. **Internal Logical Files(ILF)**: The control information or logically related data that present WITHIN the system
2. **External Interface Files(EIF)**: The control data or logical data that is referenced by the system but present in another system
3. **External Inputs(EI)**:Data or information that comes from outside of a system
4. **External Outputs(EO):**Data that goes out of the system
5. **External Inquiries(EQ)**: Combination of input and output resulting in data retrieval

All these parameters are then individually assessed for complexity
FP characterises the complexity of the software system and hence can be used to depict the project time and the manpower requirement. FP method is used for data processing systems, business systems like information systems. The five parameters mentioned above are also known as **information domain characteristics.**

STEPS TO CALCULATE FUNCTION POINT ARE:

1. Count the number of functions of each proposed type.
2. Compute the Unadjusted Function Points(UFP).
3. Find Total Degree of Influence(TDI)
4. Compute Value Adjustment Factor(VAF).
5. Find the Final Function Point Count(FPC).

b. Explain software design principles in detail illustrating with example

1. Avoiding Tunnel Vision

When designing software, it's essential to see beyond just the primary goal and consider the broader impact. Instead of focusing solely on "getting it done," developers should consider performance, scalability, and maintenance needs.

Example: A mobile application designed for ordering food should not only focus on completing orders but also on user experience, security of payment information, and ease of navigation.

2. Traceability to the Analysis Model

The design should be closely aligned with the analysis model, meaning that every requirement outlined during the analysis phase should be clearly mapped and addressed in the design. This alignment ensures that the final product meets all user needs.

Example: If the analysis phase for an e-commerce platform outlines a need for secure user login, then the design should include specifications for secure password storage and encryption.

3. Avoid “Reinventing the Wheel”

Rather than creating new components for commonly solved problems, developers should reuse existing, proven solutions. This saves time and leverages well-tested components.

Example: Instead of creating a new logging system from scratch, developers can integrate established libraries like Log4j in Java, which provides robust logging features out of the box.

4. Minimizing Intellectual Distance

Design should reduce the gap between the problem's real-world context and the software solution. The closer the software mirrors the actual problem, the easier it is to understand and maintain.

Example: In a hotel booking application, the design should represent real-world concepts like “Room,” “Guest,” and “Reservation” directly. Creating complex abstractions for these simple entities would increase intellectual distance unnecessarily.

5. Uniformity and Integration

Design should ensure consistency and cohesion among different software components, making the system appear as a unified whole. Uniformity also promotes ease of use and maintainability.

Example: A software system with multiple modules (e.g., user authentication, data processing, and reporting) should have a consistent naming convention, error-handling strategy, and user interface style across all modules.

6. Accommodate Change

Good design anticipates change by allowing easy modification without affecting the entire system. This can be achieved by modular and loosely coupled components.

Example: A billing system may need to adapt tax rates frequently. Using a configuration file for tax rates, rather than hardcoding them, allows for easy updates without modifying the codebase.

7. Graceful Degradation

Software should handle unexpected issues or errors gracefully. Instead of crashing, the software should degrade its functionality in a way that preserves as much usability as possible.

Example: A streaming service might reduce video quality when the internet connection is slow rather than interrupting playback entirely.

8. Assessing Quality

Regular evaluations should assess the design's quality, ensuring it aligns with performance, security, and usability standards. This process helps identify potential improvements early in the development lifecycle.

Example: Before deploying a new version of a social media platform, usability testing should confirm that users find the new features intuitive and functional.

9. Reviewing to Discover Errors

Regular design reviews and peer evaluations should identify potential flaws before moving to the coding phase, reducing the likelihood of costly errors in later stages.

Example: In a banking application, design reviews might reveal security flaws in the way personal data is managed, allowing for adjustments before any code is written.

10. Design is Not Coding, and Coding is Not Design

Designing software is about creating a blueprint that addresses how to solve a problem, while coding is the implementation of that blueprint. Treating them as separate activities

allows a focus on architecture and logic in design, without being limited by code syntax and structure.

Example: A designer may outline how a user authentication system should work, including account lockout after a certain number of failed attempts. The developer will later implement this using specific code, but the design phase isn’t concerned with coding specifics like syntax.