1. What are the different design principles? What is coupling and cohesion?

2. Explain architectural design for e-commerce systems?

3. Draw the use case diagram and activity diagram for the course registration system?

4. Explain architectural design for e-commerce systems?

5. Explain how Change Control & Version Control are carried out in Software Configuration Management.

6. Write RMMM plan for Library management system.

7. Suppose you are the project manager of a large software development project. Mention at least three reasons for your project delay. What are the risks associated with project delay? Perform Risk assessment and prepare RMMM plan for the same.

8. What are the Risks associated with software Projects? How do Project Managers manage such Risks?

9. Give RMMM plan and explain steps involved in generating RMMM plan for risk management systems?

10. Explain Software configuration Management process?

11. Explain Black box Testing and White box testing?

12. Explain verification and validation testing?

13. What are the different types of maintenance?

14. Write short notes on system testing?

1.What are the different design principles? What is coupling and cohesion?

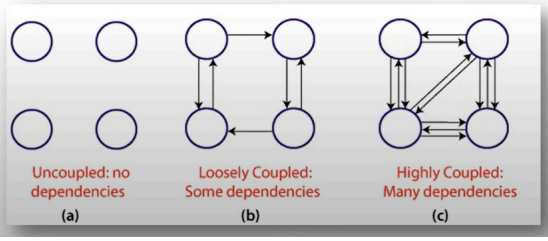
There are twelve basic principles of design: contrast, hierarchy, balance, emphasis, rhythm, unity, pattern, proportion, movement, repetition, variety and white space. These principles work together to create visual and functional designs that are meaningful to users.

Different Design Principles:(CAB RP HWU)

1. **Contrast**: Ensures a clear difference between elements (e.g., color, size, shape) to make them stand out from each other. It helps create visual hierarchy and focus.
2. **Alignment**: Arranging elements along a common line or edge to create order and organization in the design. It enhances clarity and coherence.
3. **Balance**: Refers to the distribution of visual weight in a design. It can be:
    - **Symmetrical Balance**: Elements are evenly distributed.
    - **Asymmetrical Balance**: Uneven distribution, but still feels balanced.
4. **Repetition**: Reusing the same elements (like colors, fonts, or shapes) throughout the design to create a cohesive and consistent experience.
5. **Proximity**: Grouping related elements close together to show their connection, and separating unrelated elements to avoid confusion.
6. **Hierarchy**: Arranging elements so that the most important information grabs attention first. This can be achieved through size, contrast, color, or placement.
7. **White Space (Negative Space)**: The empty space between elements, which helps improve readability and prevent the design from feeling cluttered.
8. **Unity**: Ensures all design elements are cohesive and work together to create a harmonious composition. It gives a sense of completeness to the design.

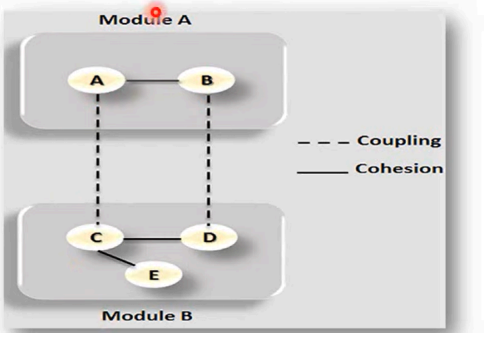
coupling

- The coupling is the degree of interdependence or number of relations between software modules.
- Two modules that are tightly coupled are strongly dependent on each other.
- However, two modules that are loosely coupled are not much dependent on each other.
- A **good design** is the one that has Low coupling.
- High coupling generates more errors because they shared large number of data.



cohesion

- Cohesion defines to the degree to which the elements of a module belong together or interrelated.
- Thus, cohesion measures the strength of relationships between pieces of functionality within a given module.
- A **good software design** will have high cohesion.



Both coupling and cohesion are important factors in determining the maintainability, scalability, and reliability of a software system. High coupling and low cohesion can make a system difficult to change and test, while low coupling and high cohesion make a system easier to maintain and improve.

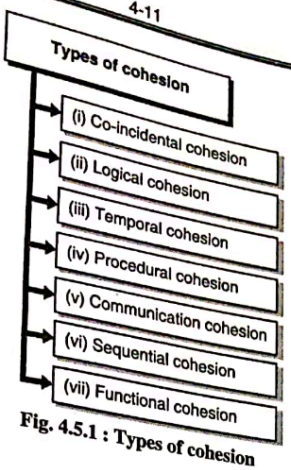


Fig. 4.5.1 : Types of cohesion

➜ **(i) Co-incidental cohesion**

- An unplanned cohesion which results into decomposition of a program into smaller components for the modularization of system is called co-incidental cohesion.
- Typically these types of cohesions are never accepted.

➜ **(ii) Logical cohesion**

- When logically classified elements are combined into a single module then it is called as logical cohesion.
- Here all elements of modules contribute to same system operation.

➜ **(iii) Temporal cohesion**

- Temporal cohesion occurs when component of a system performs more than one function and these functions must occur within same time span.
- It is generally found in initiation and termination activities.

➜ **(iv) Procedural cohesion**

- When components of system are related to each other only by sequence then there exists procedural cohesion.
- Loop in programming language is good example of procedural cohesion.

➜ **(v) Communication cohesion**

- When elements of module perform different functions but each function accepts same input and generates same output then that module is said to be communicational cohesive.
- It is sometimes acceptable if no other alternative is easily found.

➜ **(vi) Sequential cohesion**

- A module is said to be sequentially cohesive if its functions are related in a way such that output of one function acts as an input data to other function.
- This type of cohesion is easily maintainable.

➜ **(vii) Functional cohesion**

- If elements of module are grouped together since they contribute to a single function then such a module is functionally cohesive module.
- All elements of such a module are necessary for successful execution of function.

4.5.2 Advantages of Cohesion

(i) High cohesion among system components results in Better program design.

(ii) High cohesion components can be easily reused.

(iii) High cohesion components are more reliable.

4.5.3 Disadvantages of Cohesion

(i) Low cohesion components are difficult to maintain

(ii) Low cohesion components cannot be reused.

(iii) Low cohesion components are difficult to understand.

(iv) Low cohesion components are less reliable.

Syllabus Topic : Coupling

4.6 Coupling

➜ **(MU - May 15, May 16, Dec. 16, May 17, Dec. 17, May 18)**

Q. 4.6.1 What is coupling? Explain different forms of it. (Ref. sec. 4.6) **May 15, May 16, Dec. 16, May 17, Dec. 17, May 18, 5 Marks**

- Coupling is an indication of strength of interconnection between various components of a system.
- Highly coupled components are more dependent on each other whereas low - coupled components are less dependent or almost independent on each other.
- Good design always have low coupling between components of system.

Example,

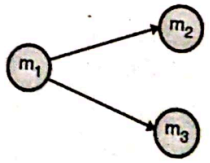


Fig. 4.6.1 : Coupling

(v) External coupling

- If two components of system share an externally imposed data format, communication protocol or device interface then they are said to be externally coupled.
- External coupling refers to communication between system components and external tools or devices.

(vi) Message coupling

- Message coupling occurs between two components of system when those components communicate with each other via message passing.
- In this coupling components are independent of each other; thus it is low type coupling.

(vii) Stamp coupling

- Stamp coupling occurs between two components of system when data between them are passed by arguments using a data structure containing elements which may or may not be used.
- Stamp coupling is also low type coupling.

4.6.2 Advantages of Coupling

(i) Low coupling components do not force ripple effect to other components.

(ii) Low coupled components can be reused.

(iii) With low coupling among components, system can built faster.

4.6.3 Disadvantages of Coupling

(i) High coupling components are difficult to understand.

(ii) High coupling components forces change in other components if one component changes.

(iii) High coupling slows down the development process.

4.6.4 Advantages of High Cohesion and Low Coupling

Q. 4.6.3 What are benefits of high cohesion and low coupling ? (Ref. sec. 4.6.4) → (MU - Dec. 17)

Dec. 17, 4 Marks

Benefits of high cohesion

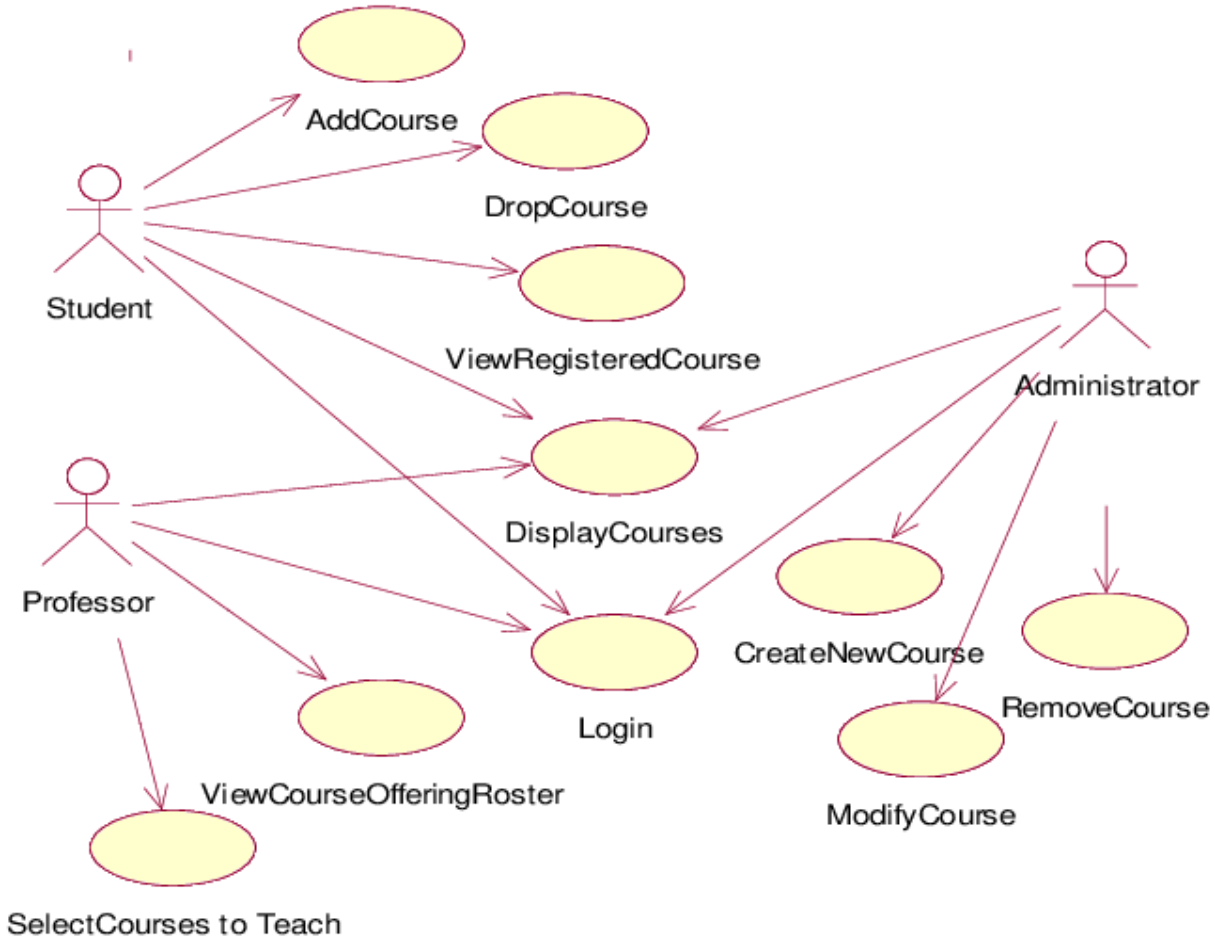
- **Readability :** (Closely) Related functions are contained in a single module.
- **Maintainability :** Debugging tends to be contained in a single module.
- **Reusability :** Classes that have concentrated functionalities are not polluted with useless functions.

Benefits of low coupling

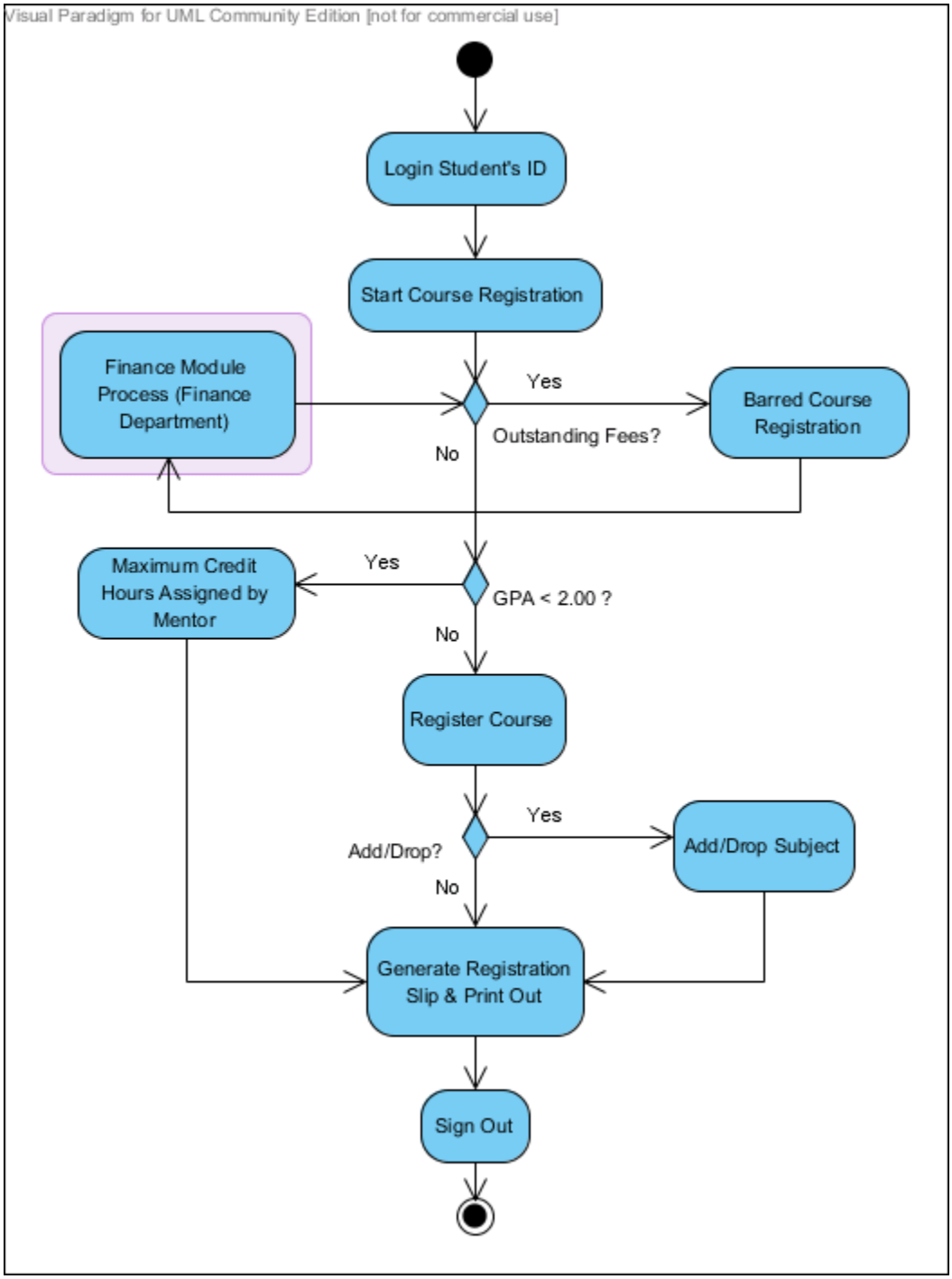
- **Maintainability :** Changes are confined in a single module.
- **Testability :** Modules involved in unit testing can be limited to a minimum.
- **Readability :** Classes that need to be analyzed are kept at a minimum.

2. Draw the use case diagram and activity diagram for the course registration system?

use case diagram:



Activity diagram:



3.Explain architectural design for e-commerce systems?

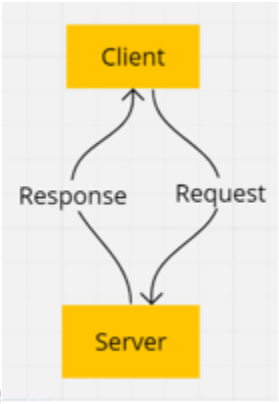
E-Commerce Architecture types

- Client-Server Architecture
- Two-Tier Architecture
- Three-Tier Architecture

Client-Server Architecture

In this architecture, the client(browser) sends the requests to the server, and the server processes the request if a request is valid then it responds with the requested data to the client. The client hosts the user interface(UI) while the server hosts the business logic and database.

- ## Two-Tier Architecture



Two-Tier Architecture

The two-tier architecture have consist of mainly two components:

1. **Client layer:** It consists of the web browser, mobile application, or the other UI that user interacts with.This front-end client makes requests to the server.
2. **Server layer:** It handels both the application logic and data storage/management.This single back-end server acts as a both the application server and the database server.

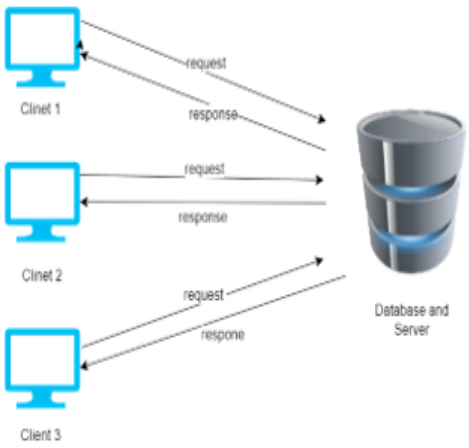
Advantages:

- It is simple to develop and deploy
- The client only communicates with one backend system
- All data logic and validation is handled on the server

Disadvantages:

- The server has to handle client requests, business logic and data storage. This can result in performance bottlenecks.

- Scalability is limited since it is not easy to scale client and data tiers independently.
- Less flexibility since presentation and data logic are coupled on the server side.



Three-Tier Architecture

The three-tier architecture is best architecture to develop a good E-commerce site. In three-tier architecture we seprates database and server that eliminate the problems we found in two-tier architecture. Three-tier architecture separates the presentation(UI), business logic and data storage layer into three distinct tiers.

- **Client tier:** Client tier is frontend layer consisting of components like a web browser, mobile application or other interfaces. This layer sends the users request and displays the response of server.
- **Middle tier:** This application server layer handles all the business logic and computational tasks. It receives requests from the client, communicates with the database to get or update data,performs calculations and other application specific tasks, and passes results back to the client.
- **Data tier:** This backend layer consists of the database servers that store and manage data. It can be a relational database like Oracle or a NoSQL database like MongoDB. The application server uses protocols like JDBC, ODBC to interact with this database tier.

Advantages:

- Separation of concerns between tiers makes application modular, flexible and easier to maintain.
- Each tier can scale independently to handle increasing loads.
- Web server can connect to multiple app servers, which in turn can connect to multiple database servers, allowing high scalability.
- Supports redundancy and failover capabilities for high availability.

Disadvantages:

- It can introduce complexity into a project. Managing three separate layers (presentation, application, and data) can be challenging, especially for small-scale applications, and it might lead to increased development and maintenance costs
- The additional layers can introduce performance overhead. Each request or transaction has to pass through the different tiers, which can slow down the system, particularly if there's a lot of data to be transferred between layers
- Scaling can be more challenging in a three-tier architecture. While it's possible to scale each layer independently, it often requires significant effort and resources to ensure that the system scales seamlessly
- Communication between layers can introduce latency in the system. When requests and responses need to traverse multiple layers, it can result in slower response times

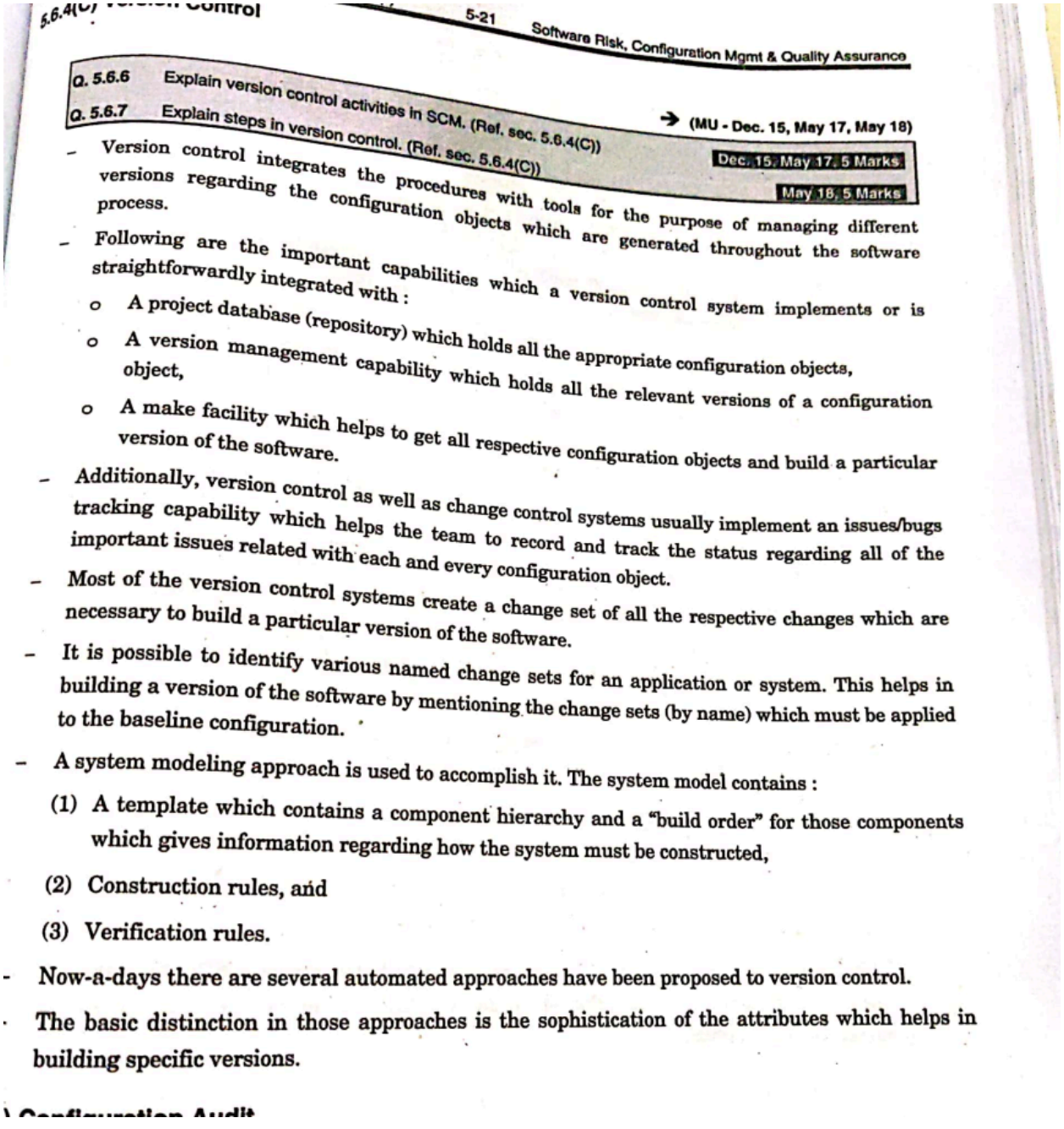
5.Explain how Change Control & Version Control are carried out in Software Configuration Management.

Software configuration management (SCM) is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced to its software products. SCM helps in identifying individual elements and configurations, tracking changes, and version selection. 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 elements: 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.



5.6.4(C) Version Control

Q. 5.6.6 Explain version control activities in SCM. (Ref. sec. 5.6.4(C))

Q. 5.6.7 Explain steps in version control. (Ref. sec. 5.6.4(C))

→ (MU - Dec. 15, May 17, May 18)

Dec. 15, May 17, 5 Marks

May 18, 5 Marks

- Version control integrates the procedures with tools for the purpose of managing different versions regarding the configuration objects which are generated throughout the software process.
- Following are the important capabilities which a version control system implements or is straightforwardly integrated with :
  - A project database (repository) which holds all the appropriate configuration objects,
  - A version management capability which holds all the relevant versions of a configuration object,
  - A make facility which helps to get all respective configuration objects and build a particular version of the software.
- Additionally, version control as well as change control systems usually implement an issues/bugs tracking capability which helps the team to record and track the status regarding all of the important issues related with each and every configuration object.
- Most of the version control systems create a change set of all the respective changes which are necessary to build a particular version of the software.
- It is possible to identify various named change sets for an application or system. This helps in building a version of the software by mentioning the change sets (by name) which must be applied to the baseline configuration.
- A system modeling approach is used to accomplish it. The system model contains :
  (1) A template which contains a component hierarchy and a "build order" for those components which gives information regarding how the system must be constructed,
  (2) Construction rules, and
  (3) Verification rules.
- Now-a-days there are several automated approaches have been proposed to version control.
- The basic distinction in those approaches is the sophistication of the attributes which helps in building specific versions.

) Configuration Audit

6.Write RMMM plan for Library management system.

RMMM Plan for Library Management System (LMS)

1. Risk Identification

1. **Misunderstanding of System Requirements:** Miscommunication between stakeholders and the development team may lead to missing or incorrect functionality.
2. **Technical Skill Shortages:** The development team may lack expertise in specific technologies required for the LMS.
3. **System Integration Issues:** Delays or incompatibility when integrating the LMS with other existing systems (e.g., financial systems or student databases) can hinder the project.
4. **Data Security Breaches:** The system could be vulnerable to cyber-attacks, compromising sensitive patron and book information.
5. **Inadequate System Scalability:** The LMS may not scale well with increased user numbers or data loads, affecting performance.
6. **Loss of Critical Data:** Hardware failure, database corruption, or lack of proper backups may lead to permanent data loss.
7. **Unreliable Internet Connectivity:** Libraries with unreliable internet connections could face issues with real-time system operations.
8. **Vendor Dependency:** Relying heavily on third-party software vendors for core functionalities may lead to issues with maintenance or future updates.
9. **User Resistance to New System:** Library staff or users might resist adopting the new system due to unfamiliarity or perceived complexity.
10. **Project Delays Due to Budget Constraints:** Budget limitations could delay procurement of necessary resources or tools, impacting development timelines.
11. **Compliance with Data Protection Laws:** The LMS must comply with laws like GDPR, and failure to do so could result in legal complications.
12. **Software Bugs and Glitches:** Development may introduce critical software bugs that could lead to system malfunctions or data integrity issues.
13. **Insufficient User Training:** Inadequate training for librarians and end users could result in improper use of the system, leading to inefficiency.
14. **Hardware Failures or Incompatibility:** Hardware malfunction or outdated infrastructure could negatively affect the system's availability and performance.
15. **Scope Creep:** Uncontrolled changes or additions to the project scope may lead to delays, budget overruns, and incomplete functionality.

---

2. Risk Mitigation

1. **Regular Stakeholder Communication:** Conduct frequent meetings with stakeholders to clarify and review requirements throughout the project lifecycle.

2. **Training and Skill Development:** Invest in developer training or hire external experts for specialized tasks where team expertise is lacking.
3. **System Testing for Compatibility:** Conduct thorough integration testing with existing systems to ensure compatibility before final deployment.
4. **Data Encryption and Security Measures:** Implement encryption, firewalls, and intrusion detection systems to mitigate data security risks.
5. **Scalable Architecture Design:** Design the LMS with scalability in mind, using cloud-based solutions that can handle increased user traffic and data loads.
6. **Regular Backups and Disaster Recovery Plan:** Ensure that regular, automated backups are in place and a disaster recovery plan is established for quick restoration of data.
7. **Offline Mode or Caching:** For unreliable internet, introduce an offline mode or data caching to ensure functionality when connectivity is lost.
8. **Avoid Vendor Lock-In:** Use open-source or widely supported platforms to avoid excessive reliance on any single vendor.
9. **Change Management Plan:** Implement a change management process, including training and support to help staff and users adapt to the new system.
10. **Regular Budget Reviews:** Conduct periodic budget reviews and adjust project plans accordingly to stay within financial constraints.
11. **Compliance Reviews:** Regularly review the LMS design and processes to ensure compliance with relevant data protection laws and regulations.
12. **Quality Assurance Testing:** Perform extensive testing throughout development to detect and resolve software bugs early in the process.
13. **Comprehensive Training Program:** Offer detailed training sessions and user manuals to ensure users are confident and efficient with the new system.
14. **Hardware Evaluation:** Regularly assess and upgrade hardware infrastructure to avoid system downtimes due to hardware issues.
15. **Scope Control:** Establish a well-defined project scope and use change control processes to manage scope changes, minimizing unnecessary additions.

---

3. Risk Monitoring

The project team will implement the following monitoring activities:

- **Weekly Risk Assessments** to review and update the risk register.
- **Regular Progress Reviews** with stakeholders to monitor requirement fulfillment and scope management.
- **Monthly Security Audits** to ensure the system remains compliant with security policies and standards.
- **Ongoing System Performance Monitoring** to identify scalability or hardware-related issues before they impact system performance.

Q7. Suppose you are the project manager of a large software development project. Mention at least three reasons for your project delay. What are the risks associated with project delay? Perform Risk assessment and prepare RMMM plan for the same

As a project manager of a large software development project, potential reasons for project delay might include:

1. **Scope Creep and Requirements Changes**: This occurs when additional features or changes to the original project requirements are introduced during development. These changes may seem small initially but can drastically alter timelines as they often require reworking previously completed code, adjusting project designs, or reallocating resources. Without proper scope management, the project can get significantly delayed as the team spends extra time accommodating these requests.
2. **Inadequate Resource Allocation and Staffing Issues**: Software projects can suffer delays when there are insufficient developers, designers, or testers to meet deadlines. Even if a team is fully staffed, delays might occur if key team members fall ill, leave the company, or become unavailable due to other project commitments. Inadequate training or expertise in critical areas can further slow down the development process, especially if specialists or consultants need to be brought in at a later stage.
3. **Integration and Technical Challenges**: Delays can arise when dealing with complex integrations between different systems, APIs, or legacy technologies, especially if unforeseen compatibility issues emerge. Technical problems such as performance bottlenecks, security vulnerabilities, or database malfunctions can also slow down development as the team must troubleshoot and fix issues. Additionally, delays may occur if the testing process uncovers significant bugs or if the architecture was not designed to scale as intended, leading to rewrites or refactoring.
4. **Inadequate Planning and Estimation**: Poor initial project planning, including inaccurate timelines and underestimating the complexity of certain features, can lead to significant delays. Without a clear breakdown of tasks, timelines can slip as developers face unexpected challenges. Misjudging the effort required for specific milestones, especially in areas like testing, deployment, or security implementation, often leads to underestimation of the time required to complete them.

5. **Poor Communication and Collaboration**: If there is a lack of clear communication among team members, stakeholders, or between different departments (e.g., development, design, QA, and management), misunderstandings can occur. This can lead to misaligned goals, rework, or overlooked issues, ultimately causing delays. In distributed teams, where communication might be asynchronous or in different time zones, these problems can be exacerbated.

Let us understand RMMM with the help of an example of high staff turnover RMMM Plan.

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.
- Organize 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**“.**

.

Q8. What are the Risks associated with software Projects? How do Project Managers manage such Risks?

Software projects are inherently complex and can face various risks that may affect their success. Here’s a look at some common risks associated with software projects and how project managers can manage them effectively.

Common Risks in Software Projects

1. Requirements Risks
    - Description: Misunderstandings or changes in requirements can lead to project scope creep or unsatisfactory deliverables.
    - Management: Employ techniques like requirements gathering workshops, regular stakeholder reviews, and agile methodologies to ensure clarity and adaptability.
2. Technical Risks
    - Description: Unfamiliar technologies, integration challenges, or architectural issues can hinder progress and lead to failures.
    - Management: Conduct thorough research and prototyping, utilize skilled personnel, and maintain updated technical documentation.
3. Resource Risks
    - Description: Insufficient resources, such as time, budget, or skilled personnel, can delay project timelines and affect quality.
    - Management: Create a realistic project plan with clear resource allocation, and monitor resource utilization regularly.
4. Schedule Risks
    - Description: Underestimating the time required to complete tasks can lead to missed deadlines.
    - Management: Use techniques like historical data for estimation, task prioritization, and buffer time for critical tasks.
5. Quality Risks
    - Description: Low-quality deliverables can result from rushed development or inadequate testing processes.

    - Management: Implement continuous testing, code reviews, and quality assurance practices throughout the project lifecycle.
6. Stakeholder Risks
    - Description: Lack of stakeholder engagement or conflicting interests can lead to misunderstandings and dissatisfaction.
    - Management: Maintain regular communication, provide updates, and involve stakeholders in key decisions.
7. Market Risks
    - Description: Changes in market conditions or competitor actions can affect project relevance or success.
    - Management: Conduct market research and maintain flexibility to pivot based on market feedback.
8. Operational Risks
    - Description: Risks arising from operational factors, such as organizational changes or policy updates, can disrupt project execution.
    - Management: Regularly review and align project goals with organizational objectives and policies.

Risk Management Strategies

1. Risk Identification
    - Conduct brainstorming sessions, surveys, and expert interviews to identify potential risks at the beginning of the project and throughout its lifecycle.
2. Risk Assessment
    - Analyze identified risks to determine their likelihood and potential impact. This can be done using qualitative and quantitative methods.
3. Risk Prioritization
    - Prioritize risks based on their severity and likelihood of occurrence to focus on the most critical ones.
4. Risk Mitigation Planning
    - Develop strategies to reduce or eliminate risks, including contingency plans, risk avoidance, and risk transfer (e.g., outsourcing).
5. Monitoring and Review
    - Continuously monitor risks throughout the project and adapt strategies as necessary. Use risk management tools to track and report on risk status.
6. Communication
    - Maintain open communication channels among team members and stakeholders about potential risks and mitigation strategies. Regularly update them on risk status and changes.
7. Documentation

- Keep detailed records of identified risks, assessments, mitigation plans, and outcomes to facilitate learning and improve future projects.

Give RMMM plan and explain steps involved in generating RMMM plan for risk management systems?

Q10. Explain Software configuration Management process?

Software Configuration Management (SCM) is a systematic process that helps manage, control, and track changes in software throughout its lifecycle. The main goals of SCM are to maintain the integrity and consistency of software products and ensure that changes are made systematically and traceably. Here's an overview of the SCM process:

1. Planning

- **Establish SCM Policies**: Define the scope, objectives, and methodologies for configuration management.
- **Identify Configuration Items (CIs):** Determine which components (code, documentation, libraries, etc.) will be managed.
- **Define Roles and Responsibilities:** Assign team members to SCM roles, such as configuration manager, release manager, and developers.

2. Identification

- **Version Control:** Assign unique identifiers (version numbers) to CIs to track changes over time.
- **Baseline Creation:** Establish baselines that represent stable points in the project, which can be referenced in the future.

3. Change Control

- **Change Request Process:** Implement a formal process for submitting change requests (CRs), including description, justification, and impact analysis.

- **Impact Analysis**: Assess the impact of proposed changes on the project scope, schedule, and resources.
- **Review and Approval:** Involve stakeholders (often through a Change Control Board) to review and approve or reject change requests.

4. Configuration Control

- **Implementation of Changes**: Once approved, changes are implemented in a controlled manner, ensuring that all modifications are properly documented.
- **Version Management**: Use version control tools (e.g., Git, SVN) to manage code changes, branches, and merges.

5. Status Accounting

- **Tracking Changes:** Maintain records of all changes, including version numbers, change requests, and current status of each configuration item.
- **Reporting**: Generate reports that provide visibility into the current state of CIs and recent changes.

6. Audit and Review

- **Configuration Audits:** Conduct regular audits to verify that the CIs match the documented specifications and that changes have been implemented correctly.
- **Compliance Checks:** Ensure that all processes comply with organizational standards and regulatory requirements.

7. Release Management

- **Build and Release Planning:** Plan for the integration and deployment of changes into the production environment.
- **Deployment:** Manage the rollout of new versions or updates to ensure minimal disruption to users.
- **Post-Release Evaluation:** Review the deployment process and gather feedback to improve future releases.

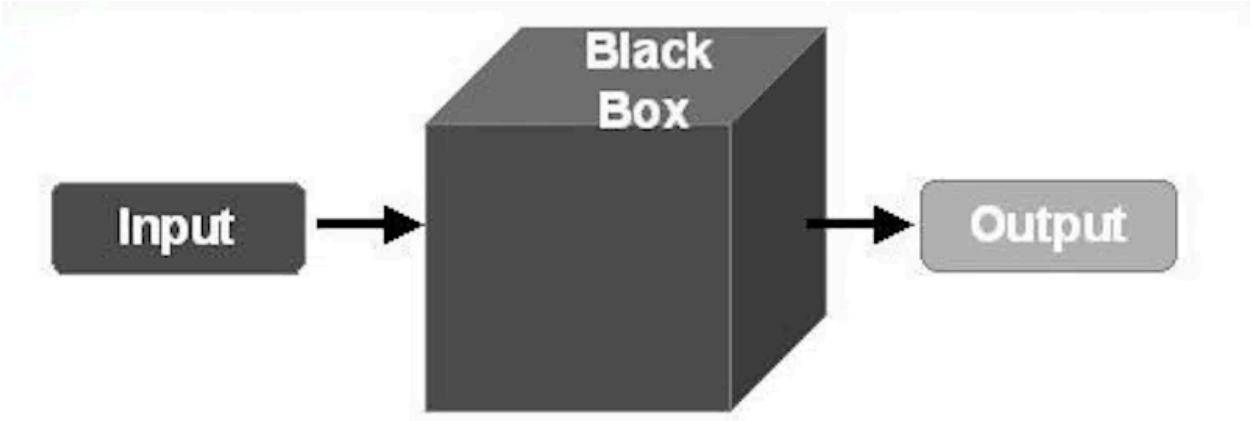
8. Maintenance

- **Ongoing Support:** Provide support for the software, addressing any issues that arise after deployment.
- **Continuous Improvement:** Adapt and refine the SCM process based on lessons learned, feedback, and changing project needs.

Q11. Explain Black box Testing and White box testing?

Black Box

Black Box Testing is an important part of making sure software works as it should. Instead of peeking into the code, testers check how the software behaves from the outside, just like users would. This helps catch any issues or bugs that might affect how the software works.Black-box testing is a type of software testing in which the tester is not concerned with the software's internal knowledge or implementation details but rather focuses on validating the functionality based on the provided specifications or requirements.



Functional Testing

- Functional testing is defined as a type of testing that verifies that each function of the software application works in conformance with the requirement and specification.
- This testing is not concerned with the source code of the application. Each functionality of the software application is tested by providing appropriate test input, expecting the output, and comparing the actual output with the expected output.
- This testing focuses on checking the user interface, APIs, database, security, client or server application, and functionality of the Application Under Test. Functional testing can be manual or automated. It determines the system's software functional requirements.

Regression Testing

- Regression Testing is the process of testing the modified parts of the code and the parts that might get affected due to the modifications to ensure that no new errors have been introduced in the software after the modifications have been made.

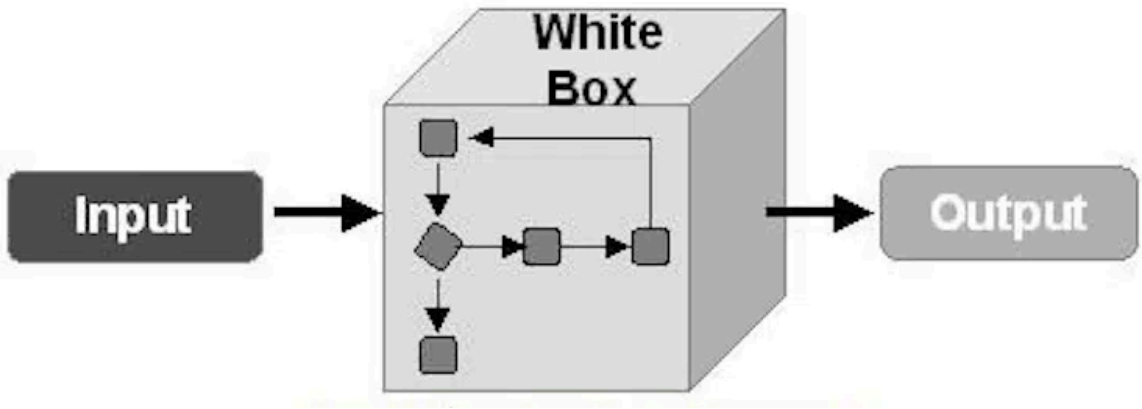
- Regression means the return of something and in the software field, it refers to the return of a bug. It ensures that the newly added code is compatible with the existing code.
- In other words, a new software update has no impact on the functionality of the software. This is carried out after a system maintenance operation and upgrades.

Nonfunctional Testing

- Non-functional testing is a software testing technique that checks the non-functional attributes of the system.
- Non-functional testing is defined as a type of software testing to check non-functional aspects of a software application.
- It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.
- Non-functional testing is as important as functional testing.
- Non-functional testing is also known as NFT. This testing is not functional testing of software. It focuses on the software's performance, usability, and scalability.

White box testing

White box testing techniques analyze the internal structures, the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing clear box testing or structural testing. White Box Testing is also known as transparent testing or open box testing. 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. White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.



Unit Testing

- Checks if each part or function of the application works correctly.
- Ensures the application meets design requirements during development.

Integration Testing

- Examines how different parts of the application work together.
- Done after unit testing to make sure components work well both alone and together.

Regression Testing

- Verifies that changes or updates don’t break existing functionality.
- Ensures the application still passes all existing tests after updates.

Q12. Explain verification and validation testing?

Verification and Validation is the process of investigating whether a software system satisfies specifications and standards and fulfills the required purpose.

Verification Testing

- Definition: The process of evaluating work products (requirements, design, code) to ensure they meet specified requirements during various stages of development.
- Purpose: To ensure that the software is built correctly according to the defined specifications.
- Focus: Answers the question: "Are we building the product right?"
- Phases: Conducted during requirements gathering, design, coding, and pre-release.
- Key Activities:
    - Reviews: Systematic examination of documents and artifacts by stakeholders.
    - Inspections: Formal reviews to examine code or documents for defects.
    - Walkthroughs: Collaborative reviews involving developers and stakeholders.
    - Static Analysis: Analyzing code without execution to find potential issues.
- Outcome: Identifies issues early in the development cycle, reducing costs and time associated with fixing defects later.

Validation Testing

- Definition: The process of evaluating the final product to ensure it meets business needs and user requirements.
- Purpose: To ensure that the software fulfills its intended purpose and meets user expectations.
- Focus: Answers the question: "Are we building the right product?"
- Phases: Conducted primarily during the testing phase, typically at the end of the development cycle.
- Key Activities:
    - Functional Testing: Ensuring the software functions as intended according to requirements.
    - User Acceptance Testing (UAT): End-users validate the software in a real-world environment.
    - Performance Testing: Assessing system performance under various conditions.
    - Regression Testing: Verifying that new changes do not negatively impact existing functionality.
- Outcome: Validates that the software is user-friendly, functional, and meets all specified requirements, providing confidence to stakeholders before deployment.

Q13. What are the different types of maintenance?

Software Maintenance refers to the process of modifying and updating a software system after it has been delivered to the customer. This involves fixing bugs, adding new features, and adapting to new hardware or software environments. Effective maintenance is crucial for extending the software's lifespan and aligning it with evolving user needs. It is an essential part of the software development life cycle (SDLC), involving planned and unplanned activities to keep the system reliable and up-to-date.

1. Corrective Maintenance

- **Description:** Performed after a fault or failure occurs to restore the system to its normal operational state.
- **Importance:** Essential for addressing unexpected issues that disrupt functionality, ensuring quick recovery and minimal downtime.
- **Examples:**
    - **Software Bugs:** Fixing a critical bug in a library management system that prevents users from checking out books.
    - **Hardware Failures:** Replacing a failed hard drive in a server that has caused data loss or system crashes.

2. Preventive Maintenance

- **Description:** A proactive approach involving regular checks and maintenance activities to prevent potential issues before they arise.
- **Importance:** Extends the lifespan of the system and reduces the likelihood of unexpected failures through routine inspections and updates.
- **Examples:**
    - **Software Updates:** Regularly updating a content management system to patch vulnerabilities and improve performance.
    - **Hardware Inspections:** Conducting periodic checks on servers and networking equipment to ensure they are functioning properly and to replace any worn-out components.

3. Predictive Maintenance

- **Description:** Utilizes data analysis and monitoring to predict when maintenance should be performed based on actual performance and system condition.
- **Importance:** Optimizes maintenance schedules, reduces costs, and prevents breakdowns by addressing issues before they escalate.
- **Examples:**
    - **Monitoring System Performance:** Using tools to track CPU usage and memory allocation, allowing for maintenance before performance issues arise.

- **Vibration Analysis:** In a manufacturing facility, monitoring the vibrations of machinery to predict when parts might fail and need replacement.

4. Adaptive Maintenance

- **Description:** Involves making modifications to the system to adapt to changes in requirements or operational environments.
- **Importance:** Ensures that the system remains relevant and effective in meeting current user needs, keeping software and hardware up to date with industry standards and regulations.
- **Examples:**
  - **Regulatory Compliance Updates:** Modifying an accounting software to comply with new tax laws or financial regulations.
  - **Feature Enhancements:** Adding new functionalities to an online library system based on user feedback, such as integrating e-book lending capabilities.

5. Emergency Maintenance

- **Description:** Immediate action taken to address unexpected failures or critical issues requiring urgent attention.
- **Importance:** Critical for minimizing downtime and restoring services quickly in case of severe system malfunctions, ensuring operational continuity.
- **Examples:**
  - **Security Breaches:** Responding to a data breach in a library management system by quickly applying patches and securing sensitive information.
  - **System Outages:** Addressing a complete server failure that disrupts access to an online library database, requiring rapid troubleshooting and repair to restore access.

Q14. Write short notes on system testing?

System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users. This type of testing is performed after the integration testing and before the acceptance testing.

Types of system Testing

- Performance Testing: Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.

- Load Testing: Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- Stress Testing: Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- Scalability Testing: Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Tools used for System Testing

- JMeter
- Gallen Framework
- HP Quality Center/ALM
- IBM Rational Quality Manager
- Microsoft Test Manager

Advantages

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing

Disadvantages

- This testing is time consuming process than another testing techniques since it checks the entire product or software.
- The cost for the testing will be high since it covers the testing of entire software.
- It needs good debugging tool otherwise the hidden errors will not be found.
- Can be time-consuming and expensive.
- Requires adequate resources and infrastructure.
- Can be complex and challenging, especially for large and complex systems.