1. Explain Agile Process Model.
2. Differentiate b/w White Box and Black Box Testing
3. What is Cost Estimation? Explain LOC Method.

Cost estimation is the process of predicting the cost of a software project. It is an essential aspect of software project management, as accurate cost estimation helps in effective budgeting and resource allocation. One of the methods used for cost estimation in software development is the Lines of Code (LOC) method.

The LOC method is a technique that estimates the cost of a software project based on the number of lines of code (LOC) that will be written. The method involves the following steps:

Determine the size of the project: The first step is to estimate the size of the software project in terms of the number of lines of code that will be written. This can be done by reviewing the project requirements and specifications.

Determine the cost per line of code: Once the size of the project has been estimated, the next step is to determine the cost per line of code. This can be done by analyzing historical data or industry benchmarks for similar projects.

Multiply the size of the project by the cost per line of code: The final step is to multiply the estimated size of the project by the cost per line of code to get the estimated cost of the project.

The LOC method has several advantages, including:

Simplicity: The LOC method is simple and easy to understand, making it a popular choice for cost estimation.

Accuracy: The LOC method can be accurate if the project size and cost per line of code are estimated correctly.

However, the LOC method also has some disadvantages, including:

Dependency on lines of code: The LOC method relies on the number of lines of code to estimate the project cost, which may not always be a good measure of complexity or effort.

Lack of consideration for other factors: The LOC method does not consider other factors that can impact project cost, such as team experience, technology, and project scope.

In summary, the LOC method is a straightforward technique for cost estimation in software development that relies on the number of lines of code. While it has its advantages, it is important to consider other factors and use multiple estimation techniques for more accurate cost predictions.

1. List the principles of Software Design.
2. What is Change Control? How it is different from version control.

Change control and version control are two important concepts in software engineering that are often confused with each other. Here's a brief explanation of each:

Change Control:

Change control is the process of managing changes to software artifacts throughout the software development lifecycle. It involves the identification, documentation, review, approval, and implementation of changes to software artifacts such as requirements, design documents, source code, test cases, and other project deliverables. Change control ensures that changes are made in a controlled manner, with proper documentation and approvals, to minimize the risk of errors, conflicts, and delays.

Change control is an important process for maintaining the quality and integrity of software projects, as it allows project teams to manage changes systematically and to track the impact of changes on project deliverables.

Version Control:

Version control, on the other hand, is the process of managing changes to source code and other software artifacts by keeping track of different versions or revisions of the code or document. Version control systems allow developers to collaborate on software development projects, maintain a history of changes, and track who made what changes and when.

Version control is typically used for software source code, but can also be used for other software artifacts such as design documents, test cases, and user manuals.

Difference between Change Control and Version Control:

The main difference between change control and version control is that change control is a process for managing changes to all software artifacts, while version control is a process for managing changes to specific software artifacts such as source code.

Change control focuses on the process of managing changes, from identification to implementation, while version control focuses on the history and management of different versions or revisions of a specific software artifact.

In summary, while both change control and version control are important concepts in software engineering, they are different processes that serve different purposes. Change control manages changes to all software artifacts, while version control manages changes to specific artifacts such as source code.

1. Describe boundary value analysis with suitable examples.

Boundary value analysis is a software testing technique that involves testing the boundaries or edge values of input parameters to identify potential defects or issues in software applications. This technique is based on the assumption that defects often occur at the boundaries or limits of the input values.

Here are some examples of boundary value analysis with suitable examples:

1. Input parameter range:

Suppose a software application requires a user to enter a number between 1 and 1000. To perform boundary value analysis, we would test the input values of 1, 2, 999, and 1000, as these values are at the boundaries of the input range. Additionally, we might also test values just outside the range, such as 0, 101, and 1001.

2. Date and Time:

Suppose a software application requires a user to enter a date between 01/01/2000 and 12/31/2023. To perform boundary value analysis, we would test the input values of 01/01/2000, 12/31/2023, and values just outside the range, such as 12/31/1999 and 01/01/2024.

3. Currency values:

Suppose a software application requires a user to enter a currency value between $1 and $10,000. To perform boundary value analysis, we would test the input values of $1, $10,000, and values just outside the range, such as $0.99 and $10,001.

4. Password length:

Suppose a software application requires a user to create a password with a length of 8 to 12 characters. To perform boundary value analysis, we would test passwords with lengths of 8, 9, 11, and 12 characters, as well as passwords just outside the range, such as 7 and 13 characters.

By testing the boundaries and edge values of input parameters using boundary value analysis, software developers can identify defects or issues that might not be identified through normal testing. This technique can help to improve the quality and reliability of software applications.

1. Develop a SRS for Hospital \*\*\*
2. Explain Coupling and Cohesion.
3. Explain different types of testing.

There are several types of testing that software development teams use to ensure the quality, functionality, and usability of software applications. Here are some of the most common types of testing:

Unit Testing:

Unit testing is the process of testing individual code units or components in isolation to verify that they work correctly. It is typically performed by developers and involves writing and executing test cases for individual functions or methods.

Integration Testing:

Integration testing is the process of testing how different components of a software application work together. It is typically performed after unit testing and involves testing the integration between modules or subsystems to verify that they work correctly as a whole.

System Testing:

System testing is the process of testing the entire software application to ensure that it meets the functional and non-functional requirements. It is typically performed by a separate testing team and involves testing the software application as a whole in a simulated environment.

Acceptance Testing:

Acceptance testing is the process of testing the software application to ensure that it meets the customer's requirements and is ready for deployment. It is typically performed by the customer or end-user and may involve manual or automated testing.

Regression Testing:

Regression testing is the process of re-testing software applications after changes or updates have been made to ensure that previously working functionality has not been affected. It is typically performed after system testing and may involve manual or automated testing.

Performance Testing:

Performance testing is the process of testing the software application to ensure that it can handle a large number of users or transactions without performance issues. It is typically performed by a separate testing team and involves measuring and analyzing the performance of the software application under different loads.

Security Testing:

Security testing is the process of testing the software application to ensure that it is secure and cannot be easily hacked or compromised. It is typically performed by a separate testing team and may involve manual or automated testing.

By using a combination of these testing types, software development teams can ensure that software applications are of high quality, reliable, and meet the needs of customers and end-users.

1. Explain Software configuration process.

Software configuration management (SCM) is the process of identifying, organizing, and controlling changes to software applications. It involves managing the configuration of software artifacts, such as code, documentation, and other project deliverables, to ensure that changes are properly tracked and managed throughout the software development lifecycle.

Here are the steps involved in the software configuration management process:

Configuration Identification:

The first step in the software configuration management process is to identify the software artifacts that need to be managed. This involves creating a baseline for the software configuration and identifying the versions of software artifacts that will be tracked.

Configuration Control:

The second step is to control changes to the software configuration. This involves establishing a change control process that identifies who can request changes, how changes will be evaluated and prioritized, and how changes will be implemented.

Configuration Status Accounting:

The third step is to maintain a record of the status of software artifacts. This involves tracking changes to the software configuration and documenting the current state of software artifacts, including their versions, release dates, and associated documentation.

Configuration Auditing:

The fourth step is to audit the software configuration to ensure that it is consistent with the software development plan and any relevant policies or standards. This involves reviewing the configuration items and verifying that they have been properly identified, controlled, and documented.

Configuration Reporting:

The fifth step is to report on the status of the software configuration. This involves generating reports that show the current state of software artifacts, such as change requests, change implementation status, and software versions.

Build and Release Management:

The final step is to manage the software build and release process. This involves building the software from the configuration items and releasing it to customers or end-users. This step ensures that the software is delivered in a controlled and consistent manner.

By following the software configuration management process, software development teams can manage changes to software applications, maintain a record of the software configuration, and ensure that software artifacts are properly tracked and controlled throughout the software development lifecycle.

1. What are the different types of risk?

In the context of software development, there are several types of risks that can impact a project. Here are some of the most common types of risks:

1. Technical Risk:

Technical risks refer to the risks associated with the software development process itself. These risks can include problems with coding, design, integration, and testing. Technical risks can be mitigated through careful planning, testing, and implementation.

2. Schedule Risk:

Schedule risks refer to the risks associated with completing a project on time. These risks can include missed deadlines, delays in the development process, and unexpected changes to the project scope. Schedule risks can be mitigated through effective project planning, risk management, and communication with stakeholders.

3. Cost Risk:

Cost risks refer to the risks associated with project budget and expenses. These risks can include unexpected costs associated with software development, changes to project scope, and problems with resource allocation. Cost risks can be mitigated through careful budget planning, cost tracking, and resource management.

4. Business Risk:

Business risks refer to the risks associated with the impact of the software on the business or organization. These risks can include changes in the competitive landscape, changes in customer needs or expectations, and unforeseen market conditions. Business risks can be mitigated through careful market analysis, customer feedback, and ongoing communication with stakeholders.

5. Security Risk:

Security risks refer to the risks associated with the security of the software application. These risks can include security vulnerabilities, data breaches, and other security threats. Security risks can be mitigated through careful planning, testing, and ongoing security monitoring and maintenance.

6. Legal Risk:

Legal risks refer to the risks associated with compliance with legal and regulatory requirements. These risks can include issues related to intellectual property, privacy, and data protection. Legal risks can be mitigated through careful compliance planning, ongoing monitoring of legal and regulatory changes, and proactive risk management.

By identifying and managing these different types of risks, software development teams can mitigate potential problems and ensure the successful delivery of software projects.

1. Explain reverse engineering.

Reverse engineering is the process of analyzing an existing product, system or software application to understand its design, architecture, and functionality. It involves breaking down the product or software application into its constituent components, examining its behavior and structure, and then creating a new version of the product or software application based on this information.

The reverse engineering process typically involves the following steps:

1. Analyzing the Existing Product:

The first step in reverse engineering is to analyze the existing product or software application to determine its behavior and structure. This involves examining the product's components, software code, and functionality to understand how they work together.

2. Breaking Down the Product:

The second step is to break down the product into its constituent parts. This may involve disassembling hardware components, examining software code, and identifying individual functions and modules.

3. Examining the Product's Behavior:

The third step is to examine the product's behavior under various conditions. This may involve testing the product or software application under different scenarios to understand its performance and functionality.

4. Creating a Design Specification:

The fourth step is to create a design specification based on the information gathered in the previous steps. This may involve creating new software code, designing new hardware components, or creating a new product design based on the existing product.

5. Creating a New Version:

The final step is to create a new version of the product or software application based on the design specification. This may involve creating new software code, designing new hardware components, or creating a new product design based on the existing product.

Reverse engineering can be used for a variety of purposes, including product design, software development, and security analysis. It can be a useful tool for understanding how existing products and software applications work, and for creating new versions based on this information. However, reverse engineering can also raise legal and ethical concerns, particularly if it involves copying proprietary software or hardware designs without permission.

1. Draw the Data Flow diagram(upto 2 level) for the safe home software

Level 0 DFD:

The level 0 DFD would show the main functions of the "Safe Home" software system, including inputs and outputs. It would likely include functions such as "Monitor Home Security Status," "Alert Homeowner in Case of Security Breach," and "Control Home Security Devices." Inputs might include data from security sensors and outputs could include alerts to the homeowner or notifications to emergency services.

Level 1 DFD:

The level 1 DFD would show the detailed functions of each of the processes identified in the level 0 DFD. For example, the "Monitor Home Security Status" process might be broken down into more detailed processes such as "Monitor Door Sensor," "Monitor Window Sensor," and "Monitor Motion Sensor." Each of these processes would have specific inputs, such as sensor data, and specific outputs, such as status updates or alerts.

1. Explain Software Quality Assurance.

Software Quality Assurance (SQA) is the process of ensuring that software products and processes comply with specified requirements and standards, and that they are free from defects or errors. The objective of SQA is to ensure that the software product is of high quality, reliable, and meets the user's expectations.

The SQA process involves the following activities:

1. Defining Quality Standards: Quality standards are the basis for SQA. The standards should be defined in advance and should be clearly communicated to all members of the software development team.

2. Establishing a Quality Assurance Plan: A quality assurance plan is a document that outlines the SQA process and defines the roles and responsibilities of the members of the software development team.

3. Conducting Reviews and Audits: Reviews and audits are conducted to identify defects and errors in the software development process. This helps in identifying the root cause of the problem and enables the team to take corrective actions.

4. Testing: Testing is a critical component of SQA. It involves identifying and correcting defects in the software product.

5. Tracking and Reporting Defects: Defect tracking and reporting is an important component of SQA. It helps in tracking the defects and their status and helps in prioritizing the defects that need to be addressed first.

6. Implementing Continuous Improvement: Continuous improvement is an ongoing process in SQA. It involves identifying areas for improvement in the software development process and taking corrective actions to improve the quality of the software product.

The benefits of SQA include:

1. Increased Customer Satisfaction: SQA helps in delivering high-quality software products that meet customer requirements and expectations.

2. Reduced Costs: SQA helps in reducing the cost of software development by identifying defects early in the development process, thus reducing the cost of fixing defects at a later stage.

3. Improved Productivity: SQA helps in improving the productivity of the software development team by identifying areas for improvement in the software development process and implementing corrective actions.

4. Increased Credibility: SQA helps in increasing the credibility of the software development team by delivering high-quality software products that meet customer requirements and expectations.

1. Explain requirement elicitation in detail. Explain any two methods

Requirement elicitation is the process of discovering, analyzing, and documenting the needs and expectations of stakeholders for a software system. This is a crucial step in software development as it lays the foundation for the entire project. The goal of requirement elicitation is to identify the necessary features, functions, and constraints of the system in order to meet the stakeholders' needs and expectations.

There are several methods used for requirement elicitation, including:

Interviews: One of the most common methods for requirement elicitation is conducting interviews with stakeholders. Interviews can be conducted individually or in groups, and can be structured or unstructured. The purpose of the interview is to ask stakeholders open-ended questions to gather their perspectives on the system and its requirements.

Workshops: Workshops are another popular method for requirement elicitation. They involve bringing together stakeholders and development team members to discuss the system and its requirements. Workshops can be used to brainstorm ideas, identify user stories, and prioritize requirements.

Surveys: Surveys can be used to gather information from a large number of stakeholders in a cost-effective way. Surveys can be conducted online or in-person, and can be used to gather quantitative data on stakeholder needs and expectations.

Observations: Observations involve observing stakeholders in their natural environment to understand their needs and expectations for the system. This method can be useful for understanding how stakeholders currently perform tasks and identifying opportunities for improvement.

1. Explain Risk Assessment and risk projection.

Risk assessment is the process of identifying, analyzing, and evaluating risks associated with a software project. The objective of risk assessment is to determine the likelihood of a risk occurring and the potential impact it could have on the project. Risk projection, on the other hand, is the process of predicting future risks based on historical data and current trends.

The risk assessment process involves the following steps:

1. Risk Identification: Identify the potential risks that could impact the software project, such as technical risks, organizational risks, and project risks.

2. Risk Analysis: Analyze each identified risk to determine the likelihood of occurrence, the potential impact on the project, and the level of urgency to mitigate the risk.

3. Risk Evaluation: Evaluate each identified and analyzed risk based on its priority and the resources available to address it.

4. Risk Mitigation: Develop and implement a plan to mitigate the identified risks, which could include risk avoidance, risk transfer, risk reduction, or risk acceptance.

5. Risk Monitoring and Control: Continuously monitor the identified risks and the effectiveness of the mitigation plan, and adjust the plan as needed to ensure that the project stays on track.

Risk projection is the process of predicting future risks based on historical data and current trends. This involves analyzing past risks and their outcomes and identifying patterns and trends that can help predict future risks. Risk projection is an ongoing process that should be performed regularly throughout the project.

The benefits of risk assessment and risk projection include:

1. Improved decision-making: Risk assessment and risk projection provide valuable information that can help project managers make informed decisions about how to manage risks and allocate resources.

2. Enhanced project planning: The insights gained from risk assessment and risk projection can help project managers better plan for potential risks and avoid potential delays or cost overruns.

3. Increased stakeholder confidence: A comprehensive risk management plan that includes risk assessment and risk projection can increase stakeholder confidence in the project and its outcome.

In summary, risk assessment and risk projection are important processes that can help project managers better understand the potential risks associated with a software project and develop effective strategies to mitigate those risks.

1. Explain different architectural styles.

Architectural style describes a system category that encompasses:

a set of components (e.g., a database, computational modules) that

perform a function required by a system,

a set of connectors that enable “communication, coordination, and

cooperation” among components, constraints that define how components can be integrated to form the system, and semantic models that enable a designer to understand the overall properties of a system.

Types of architecture:

1. data centered

2. data flow

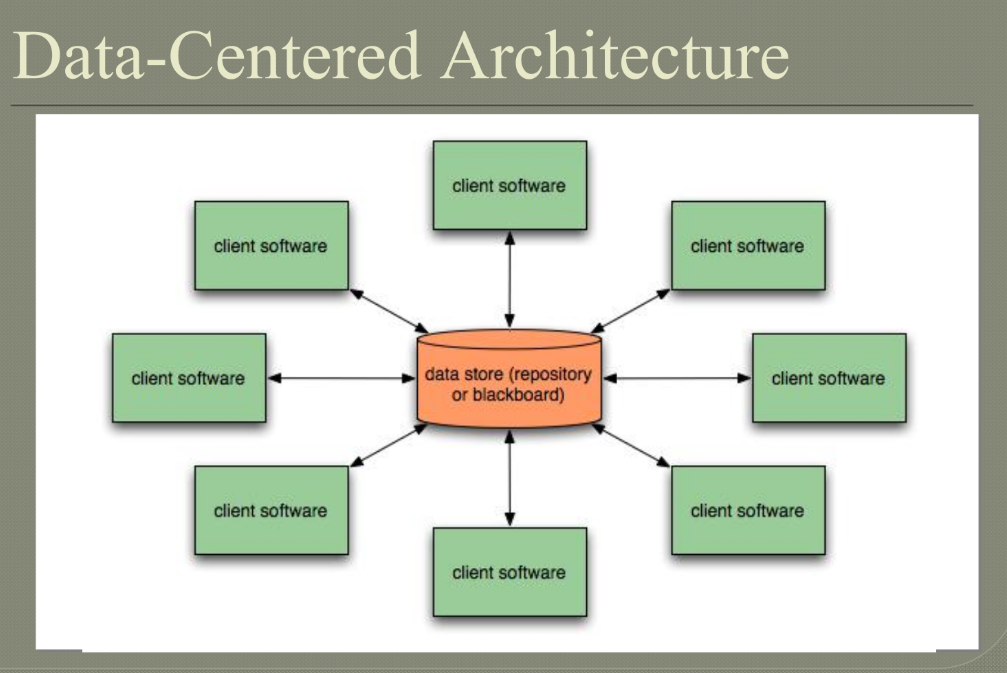
3. call and return

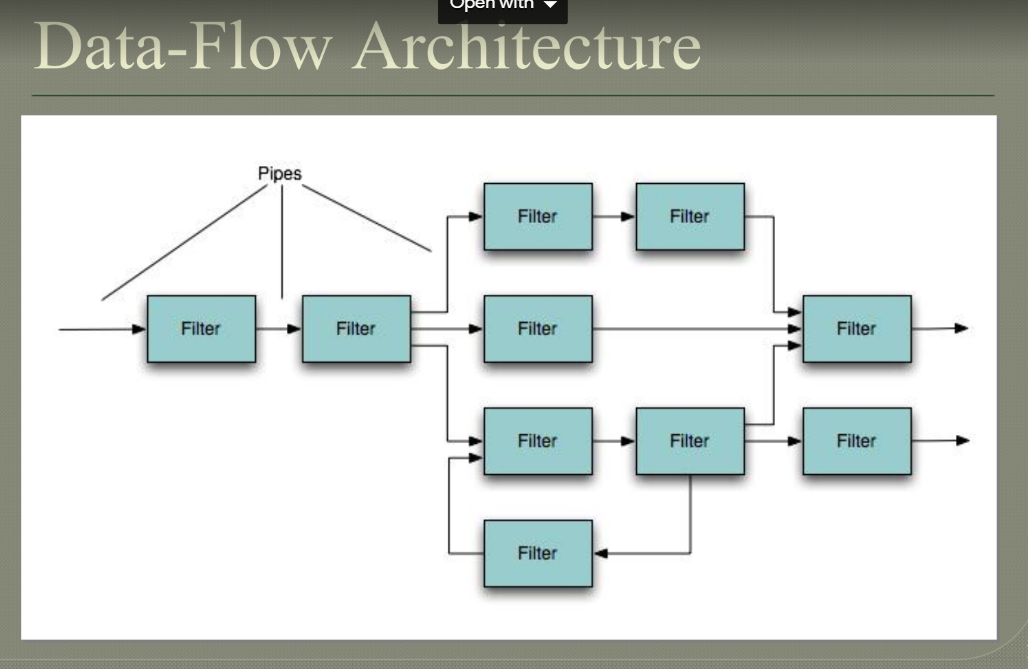
4. object oriented

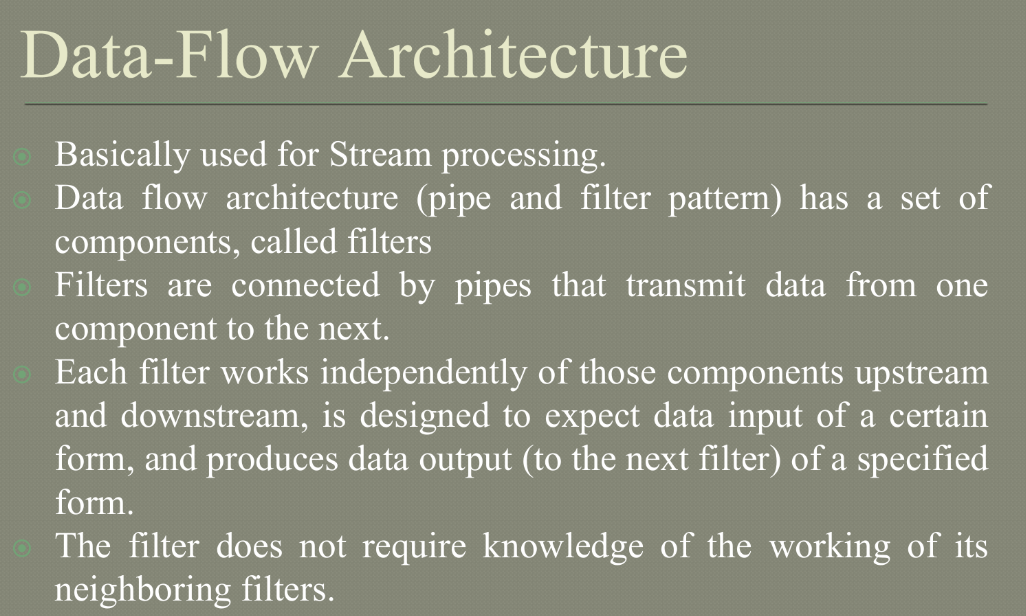
5. layered

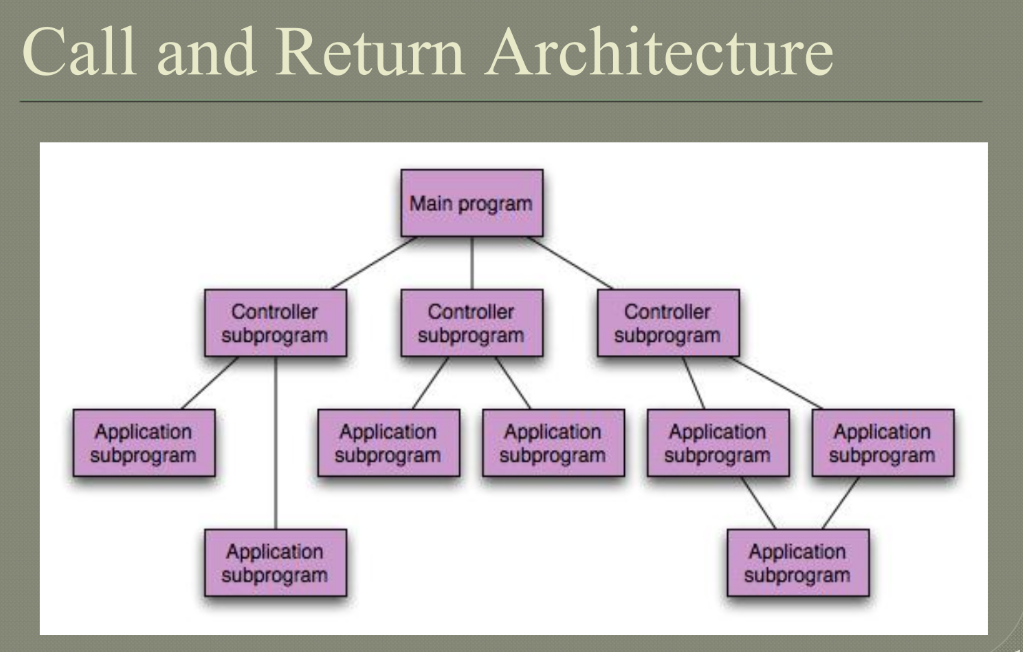
Data-centered architecture is a software architecture pattern that focuses on the storage, management, and access of data. In this architecture, data is considered the central component around which all other components are built.

In a data-centered architecture, data is stored in a central repository, such as a database, and all other components interact with the data through well-defined interfaces. The architecture is designed to ensure that data is accurate, consistent, and available to all components that need it.





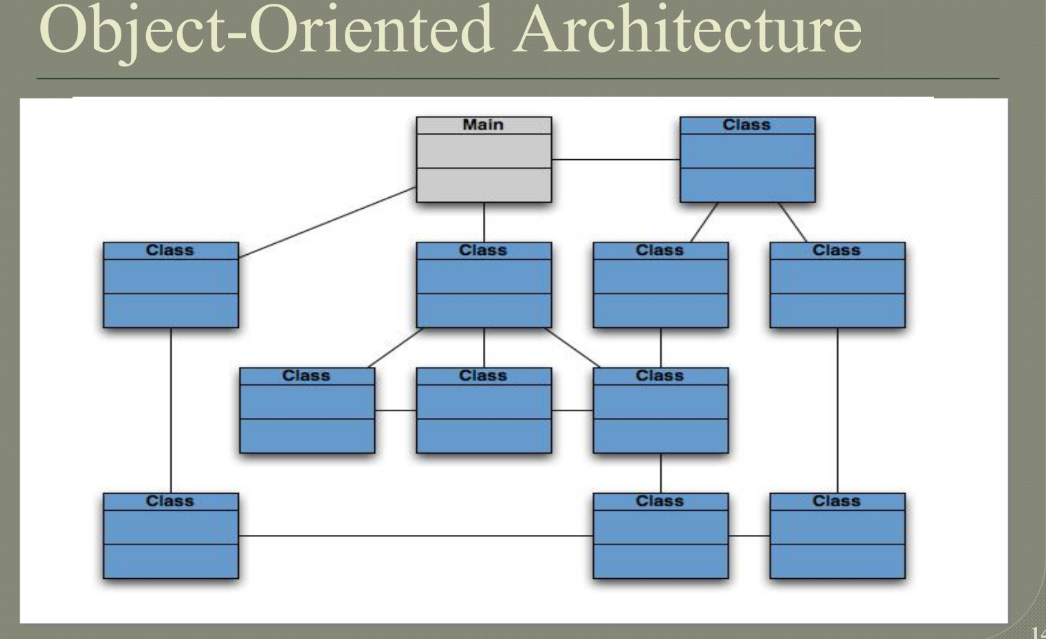




Call and return architecture, also known as subroutine architecture, is a design pattern commonly used in software engineering to structure programs into smaller, reusable components.

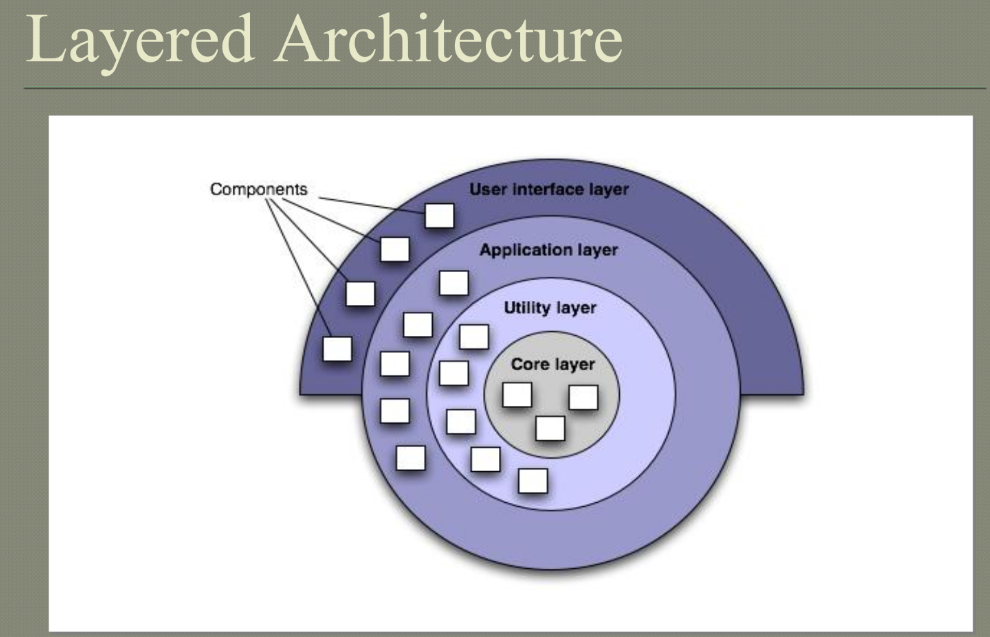
In call and return architecture, a program is divided into subroutines or procedures, each of which performs a specific task. When the program needs to perform that task, it "calls" the appropriate subroutine, passing any necessary parameters. The subroutine then executes its instructions and "returns" control back to the calling program.

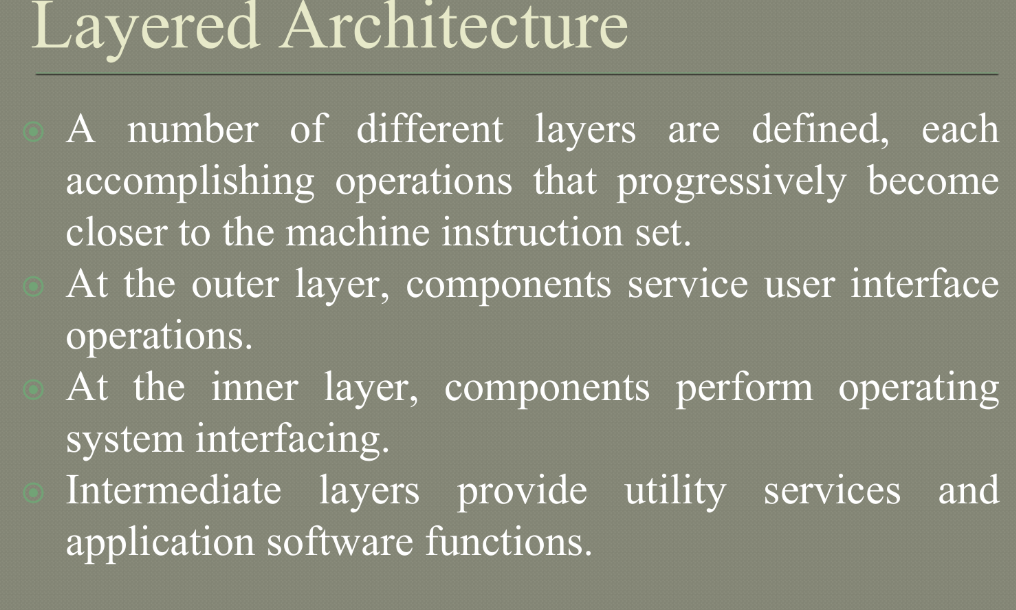




Object-oriented architecture is a design paradigm that is widely used in software engineering. It involves organizing software systems into a collection of objects that communicate with each other through messages.

In an object-oriented architecture, the system is organized into a collection of classes, which define the properties and behavior of objects. Objects are instances of classes, and communicate with each other through messages. The use of object-oriented architecture can promote modularity, flexibility, and code reuse, and is widely used in a variety of software applications.





1. Explain feasibility studies in detail.

A feasibility study is an evaluation of a proposed software project to determine whether it is technically feasible, financially viable, and operationally practical. The purpose of a feasibility study is to identify potential problems and challenges that could impact the success of the project, and to determine whether the project is worth pursuing.

The following are the main components of a feasibility study:

1. Technical Feasibility: This involves evaluating the technical aspects of the proposed software project, such as whether the necessary technology and infrastructure exist to support the project, whether the project can be completed within the required timeframe, and whether the project can be integrated with existing systems and applications.

2. Economic Feasibility: This involves evaluating the financial aspects of the proposed software project, such as the estimated costs of development, implementation, and maintenance, as well as the potential benefits and returns on investment.

3. Operational Feasibility: This involves evaluating whether the proposed software project can be implemented and integrated with existing business processes and operations, and whether it can be used effectively by end-users.

4. Legal and Regulatory Feasibility: This involves evaluating whether the proposed software project complies with relevant laws and regulations, such as data privacy laws, intellectual property laws, and industry-specific regulations.

The following steps are typically involved in conducting a feasibility study:

1. Define the problem or opportunity that the software project is intended to address.

2. Identify and document the project requirements and constraints, such as budget, timeline, and technology stack.

3. Conduct a thorough analysis of the technical, economic, operational, legal, and regulatory feasibility of the proposed project.

4. Identify potential risks and challenges associated with the proposed project, and develop contingency plans to mitigate those risks.

5. Prepare a feasibility study report summarizing the findings of the analysis, including the estimated costs and benefits of the proposed project.

The findings of a feasibility study are used to make an informed decision on whether to proceed with the proposed software project, and to develop a detailed project plan that takes into account the identified risks and challenges.

1. Explain Kanban Model.

Kanban is a software development methodology that emphasizes visual management, continuous delivery, and incremental improvement. The name "Kanban" comes from the Japanese word for "visual signal," which refers to the use of a visual board to track work in progress.

In the Kanban model, work is broken down into small, manageable pieces and each piece is represented by a card or sticky note on a Kanban board. The board is typically divided into columns representing the stages of the development process, such as "to do," "in progress," and "done." Each card is moved across the board as it progresses through the stages of the process, providing a visual representation of the work in progress.

The Kanban model has several **key principles:**

**Visual management**: The use of a visual board to track work in progress provides a clear and transparent view of the development process, helping to identify bottlenecks and improve efficiency.

**Limiting work in progress:** The number of cards allowed in each column is limited, preventing the team from taking on too much work at once and helping to maintain a steady flow of work.

**Continuous delivery:** Work is delivered in small, frequent increments, allowing the team to quickly respond to changing requirements and feedback from stakeholders.

**Continuous improvement:** The team regularly reviews and reflects on their process, identifying areas for improvement and making incremental changes to improve efficiency and quality.

The Kanban model is flexible and adaptable, making it well-suited for teams that work in a dynamic and changing environment. It is often used in Agile software development, but can be applied to other types of projects as well. The focus on visual management and continuous improvement helps teams to identify and address issues quickly, leading to a more efficient and effective development process.

**steps in kanban model**

The Kanban model is a software development methodology that emphasizes visual management, continuous delivery, and incremental improvement. Here are the typical steps in the Kanban model:

**Visualize the workflow:** The team maps out the development process and creates a visual board, such as a whiteboard or electronic tool, to represent the workflow. Each stage of the process is represented by a column on the board.

**Define work items:** Work items are broken down into small, manageable pieces and each piece is represented by a card or sticky note on the Kanban board.

**Limit work in progress:** The number of cards allowed in each column is limited, preventing the team from taking on too much work at once and helping to maintain a steady flow of work.

**Implement pull-based system:** Work is pulled into the next column only when there is available capacity, rather than being pushed from the previous column.

**Continuous delivery**: Work is delivered in small, frequent increments, allowing the team to quickly respond to changing requirements and feedback from stakeholders.

**Continuous improvement**: The team regularly reviews and reflects on their process, identifying areas for improvement and making incremental changes to improve efficiency and quality.

**Manage flow**: The team tracks the flow of work across the board, identifying and addressing bottlenecks and impediments to ensure a smooth and efficient process.

**Iterate**: The team repeats these steps, making incremental improvements to the process over time.

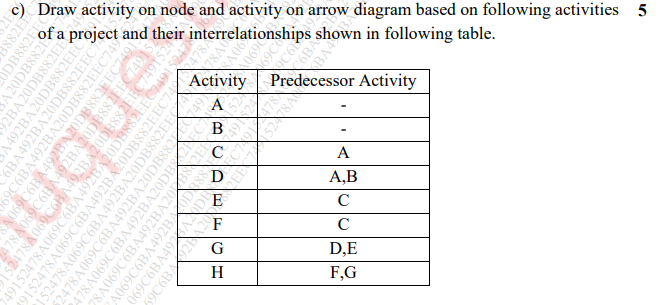
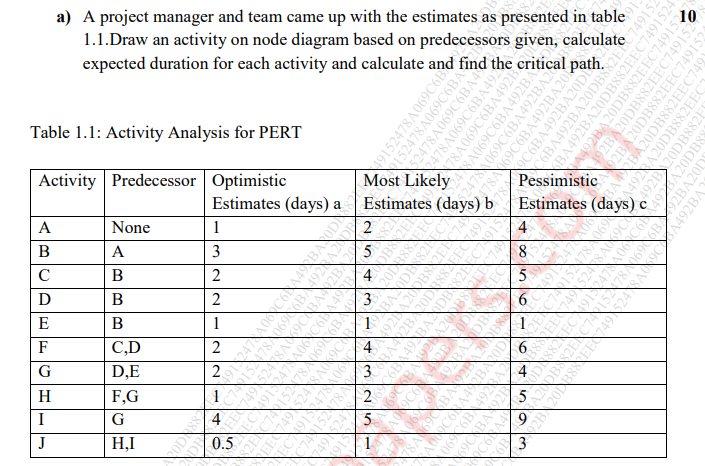
1. Short note on Process Metrics and project metrics.

Process metrics and project metrics are both important measures used in software development to assess the performance of a project or process. However, they differ in terms of what they measure and how they are used.

Process metrics are used to measure the performance of a software development process. They focus on the process itself rather than the outcome of the project. Process metrics can help identify areas for improvement in the development process, such as bottlenecks or inefficiencies. Examples of process metrics include the number of defects found during testing, the time it takes to complete a task, and the percentage of tasks completed on time.

Project metrics, on the other hand, are used to measure the success of a software development project. They focus on the outcome of the project rather than the process itself. Project metrics can help identify whether a project has met its objectives and whether it has delivered value to the customer. Examples of project metrics include the number of features delivered, the time it took to deliver the features, and the customer satisfaction rating.

Both process metrics and project metrics are important for assessing the performance of a software development project. Process metrics can help identify areas for improvement in the development process, while project metrics can help determine whether the project has met its goals. By using both types of metrics, software development teams can gain a more comprehensive understanding of their performance and identify opportunities for improvement.

1. Explain Mc Calls Quality Factor? \*\*\*\*
2. Discuss about the principles of user interface design steps?
3. Explain Evolutionary process model?
4. What are the potential problems of prototyping models?
5. What are the different steps to determine the overall consequences of risks?
6. With examples, differentiate between validation and verification.
7. What are the benefits of high cohesion and low coupling?
8. With a neat diagram explain the spiral model of software development.
9. Explain the different metrics used for software quality and reliability.
10. Explain basis path testing and cyclomatic complexity with suitable examples.
11. Explain the different OO testing methods.
12. Explain the various steps involved in change control.
13. Write short note on: SCRUM, Service Oriented Software Engineering, Schedule and Cost slippage, Security Engineering.
14. What is maintenance and different types? Explain the steps for creating the maintenance log.
15. What is FTR? Explain review guidelines considered during FTR.
16. Elaborate COCOMO method of cost estimation.
17. Explain glass path testing gin detail.
18. Explain walkthrough.
19. Discuss different categories of risk that help to define impact values in a risk table.
20. List out requirement models. Explain any one of them.
21. Explain umbrella activities of software engineering.
22. Illustrate Design Principles.
23. Differentiate between waterfall and spiral model.
24. Explain 3 P’s of software engineering.
25. Illustrate design issues.
26. Differentiate between FP based and LOC based cost estimation techniques.
27. What is the use of use case diagram? Draw use case diagram diagram for hospital management system.
28. Illustrate change control and version control.
29. Discuss functional and non-functional requirements
30. Explain the relationship among scope, schedule, and budget?
31. What is a Project? What are the attributes of a project?
32. Explain how Gantt Chart can be used for planning and controlling small projects with suitable example? What are the limitations of Gantt chart?
33. What are the advantages of including milestones in the WBS? Why should the WBS be deliverable oriented?
34. What do you mean by Project Charter and Plan? Are they different.
35. What is functional point analysis, explain in detail? What are its benefits in engineering process?
36. What are the PMBOK areas?
37. Explain in detail 4P’s of project management.
38. Differentiate between alpha and beta testing.
39. 
40. 
41. What are the objectives of requirement analysis?
42. Explain how an analysis model is translated to Design models.
43. Explain in detail the design quality guidelines and attributes.
44. What are the benefits of component based development model? With a neat diagram and explain the different phases of component based development.
45. Compare the main features and practices at each of the key phases of the agile approach and more traditional approaches of the software development life cycle.
46. Short note on test driven development.
47. Draw the Data Flow diagram up to level 2 for a restaurant management system which has online food ordering, food delivering, GST calculation, invoice creation, and payments subsystems.
48. Draw a complete use case for anyone of the following activities.

Buying a stock using an online brokerage account.

Using your charge card for a meal at a restaurant.

1. Draw the state chart diagram and activity diagram for ATM system.
2. 