CS603-2023 final term questions Answered By Pin2

Q1 refactoring and performence me diftence

the difference between refactoring and performance optimization:

Refactoring: Improves code quality and readability by restructuring code without changing its behavior.

Performance Optimization: Enhances code efficiency to make it run faster and consume fewer resources.

Q2 three types deni thi preventer change

I believe you're referring to the "Change Preventer" code smells, which are design issues in software development that can make it difficult to make changes to the codebase. Here are three types of "Change Preventer" code smells:

1. **Divergent Change**: This occurs when a single class is frequently modified for various reasons, such as adding new features, fixing bugs, or making changes unrelated to its primary responsibility. It violates the Single Responsibility Principle (SRP) and makes the codebase hard to maintain.
2. **Shotgun Surgery**: Opposite to divergent change, this smell involves making many small changes across multiple classes whenever a single change in requirements is needed. It indicates that the code is tightly coupled and lacks proper separation of concerns.
3. **Parallel Inheritance Hierarchies**: In this situation, a set of subclasses mirrors another set of classes with a parallel hierarchy. When changes

are made to one hierarchy, they often require corresponding changes in the other, leading to code duplication and maintenance challenges.

These code smells should be identified and addressed during code reviews and refactoring to improve code maintainability and extensibility.

Q3 layers se depnecies km hoti h

"In a layered architecture, dependencies between different parts of the system are minimized through clear separation of concerns, standardized interfaces, and well-defined boundaries, which allows for independent development, testing, and maintainability."

4 developer q derte hen refactoring se

"Developers may hesitate to refactor code due to concerns about potential disruptions to existing functionality, the additional time and effort required, and the need for comprehensive test coverage to ensure changes don't introduce bugs."

5 architecture design q zarori h

"Architecture design is essential in software development because it provides a structured plan for organizing and building a system. It helps ensure that the system meets functional and non-functional requirements, promotes maintainability, scalability, and performance, and facilitates team collaboration by establishing a common understanding of the system's structure and components."

Long question

1 option se diye hoy the depnecies fine krne the

2 could cuple characteristics

I apologize for the misunderstanding. Here are the fundamental characteristics of coupling in software design, irrespective of the specific types:

1. **Degree of Interdependence**: Coupling represents the level of interdependence between different modules, components, or classes within a software system.
2. **Strength of Connection**: It indicates how strongly one module relies on or influences another module.
3. **Impact of Changes**: Coupling affects how changes made to one part of the system can impact other parts. Higher coupling increases the risk of unintended consequences when making modifications.
4. **Ease of Maintenance**: Looser coupling typically leads to easier maintenance, as changes to one module are less likely to require changes in other modules.

5. **Modifiability and Flexibility**: Lower coupling enhances the system's modifiability and flexibility, allowing for easier adaptation to changing requirements.
6. **Reusability**: Loose coupling promotes reusability, as components with fewer dependencies can be more easily reused in different contexts.
7. **Testability**: Loosely coupled modules are often more straightforward to test in isolation, leading to better testability and robustness.
8. **Complexity Reduction**: Reducing coupling simplifies the codebase and makes it more understandable, aiding in code comprehension and maintenance.
9. **Parallel Development**: Looser coupling allows for parallel development of different components by different teams or developers.
10. **Scalability**: Systems with lower coupling are often more scalable, as individual components can be scaled independently.
11. **Risk Mitigation**: Minimizing coupling helps mitigate the risk of unexpected issues arising from changes made to the codebase.

These characteristics highlight the importance of managing coupling in software design to create maintainable, adaptable, and robust systems.

3 multiple view model me hun mvc se q use krte hen bht zyda asan tha paper

Using the Model-View-Controller (MVC) architectural pattern with multiple view models can be beneficial in certain scenarios. Here are solid reasons for employing MVC with multiple view models:

1. **Modularity and Separation of Concerns**: MVC promotes a clear separation of concerns. Each component (Model, View, Controller) has a distinct role, which makes it easier to manage the application's complexity. By using multiple view models, you extend this separation to the View layer, allowing for more modular and organized code.
2. **Reusability**: Multiple view models can be designed to encapsulate specific presentation logic or behaviors. These view models can be reused across different views, reducing code duplication and promoting code consistency. For example, if you have similar functionality in different parts of the application, you can reuse the appropriate view model.
3. **Testability**: MVC with multiple view models enhances the testability of your application. Each view model can be unit-tested independently of the user interface, making it easier to verify the correctness of the presentation logic.
4. **Scalability**: As your application grows, it's essential to maintain scalability. Multiple view models allow you to scale your application by adding new views

and view models without affecting the existing codebase significantly. This scalability is valuable for accommodating additional features and user interfaces.

5. **Complex User Interfaces**: In applications with complex user interfaces, using multiple view models can help manage the intricacies of different UI components. Each view model can be tailored to handle a specific portion of the interface, simplifying the code within each view model.
6. **Parallel Development**: MVC allows different teams or developers to work on different aspects of the application concurrently. One team can focus on developing view models while another works on the user interface. This parallel development can speed up the development process.
7. **Flexibility**: Multiple view models provide flexibility in adapting to changing requirements. When new views or functionality are added, you can create dedicated view models to handle them, ensuring that existing code remains unaffected.
8. **User Experience Customization**: If your application offers different user experiences or themes, each view model can be designed to support specific customization options. This approach allows for more tailored user experiences while maintaining code separation.
9. **Maintainability**: The separation of concerns facilitated by MVC and multiple view models makes it easier to maintain the application over time. Changes or updates in one area of the application are less likely to introduce unintended side effects in other parts.
10. **Consistency**: Using multiple view models can help maintain a consistent user experience throughout the application. Each view model can enforce specific standards and behaviors, ensuring uniformity in how data is presented and interacted with.

In summary, employing MVC with multiple view models is a powerful approach for building maintainable, scalable, and modular applications, especially when dealing with complex user interfaces and diverse functionality. It enables efficient code organization, reusability, and testability, which are essential for robust software development.

1)
Write Steps of polymorphism when we apply replacement in the code.

2)
Write different between refactoring and performance optimization?

3)
Scenario was given where agile methodology used.
Write Four architectural principles on agile methodologies

4)
What are the issues of refactorings?

5)

Write the characteristics of cloud computing?

6)

Which architecture style is used to achieve performance,security goals.

7) Enlisted two component and connector type patterns.

8)Enlist four quality attributes of concurrently.

9)
Write the five categories of bad smells.also tells the change preventer category steps.

1. **Polymorphism Steps with Replacement**:
    - Identify a base class or interface.
    - Create derived classes that inherit from the base class or implement the interface.
    - Override or implement methods in the derived classes, providing specialized behavior.
    - Use the base class or interface type to reference objects of the derived classes, allowing for polymorphic behavior during runtime.
2. **Difference between Refactoring and Performance Optimization**:
    - Refactoring is the process of improving the structure and design of code without changing its external behavior. It focuses on readability, maintainability, and code quality.
    - Performance optimization involves making changes to code to improve its runtime efficiency, often at the cost of code readability or maintainability.
3. **Four Architectural Principles in Agile Methodologies**:
    - **Simplicity**: Keep the architecture and design as simple as possible to minimize unnecessary complexity.
    - **Incremental Development**: Build the architecture incrementally in response to evolving requirements and feedback.
    - **Collaborative Design**: Involve team members in architectural decisions and design to encourage collaboration.

- **Embrace Change**: Be flexible and ready to adapt the architecture as requirements change over time.

4. **Issues of Refactorings**:
   - Rounding-off errors.
   - Loss of comments or documentation.
   - Introduction of new bugs if not thoroughly tested.
   - Potential impact on external interfaces and dependencies.
5. **Characteristics of Cloud Computing**:
   - On-Demand Self-Service
   - Broad Network Access
   - Resource Pooling
   - Rapid Elasticity
   - Measured Service
6. **Architecture Style for Performance and Security Goals**:
   - Service-Oriented Architecture (SOA) is often used to achieve both performance and security goals by decomposing complex systems into loosely coupled, reusable services. These services can be individually optimized for performance and security.
7. **Two Component and Connector Type Patterns**:
   - Publish-Subscribe (or Observer) Pattern
   - Client-Server Pattern
8. **Four Quality Attributes of Concurrency**:
   - **Concurrency**: The ability of the system to handle multiple tasks simultaneously.
   - **Parallelism**: The ability to execute multiple tasks in parallel for improved performance.
   - **Synchronization**: Ensuring that concurrent tasks coordinate and communicate correctly.
   - **Deadlock Avoidance**: Strategies to prevent deadlock situations in concurrent systems.
9. **Five Categories of Bad Smells and Change Preventer Category Steps**:
   - Categories: Bloaters, Object-Orientation Abusers, Change Preventers, Dispensables, and Couplers.
   - Change Preventers: These bad smells hinder code modification. Steps include identifying code that is hard to change, extracting complex methods, and reducing unnecessary code duplication.

Q#1 : Which of the following are possible refactorings applied on duplicated code?

• Extract method
• Move method
• Extract class
• Abstract class
• Pull up method
• Inherited

The possible refactorings applied on duplicated code are:

1. Extract method
2. Move method
3. Extract class
4. Pull up method

These refactorings help reduce code duplication and improve code maintainability by promoting reusability and organization.

Q#2 : how we can make refactoring safe
Q# 3 writes benefits of navigations structure in web applications
Q#4 writes benefits of testing in application
Q# 5 write types of abstract architecture
Q#6 write two types of deployment model in cloud computing
Q# 7 what reason to redesign of application.
Answers

Q#2: How to Make Refactoring Safe:

- Ensure you have comprehensive automated test coverage to catch regressions.
- Refactor in small, incremental steps rather than large, risky changes.
- Use version control systems to track changes and revert if necessary.
- Involve code reviews to get feedback from peers.
- Refactor during periods of low development activity or with proper planning.

Q#3: Benefits of Navigation Structure in Web Applications:

- Enhances user experience and usability.
- Helps users find information quickly.
- Supports easy navigation between different sections.
- Improves content organization and hierarchy.
- Facilitates consistent and intuitive user interactions.

Q#4: Benefits of Testing in Applications:

- Identifies and prevents bugs and errors.
- Ensures software meets functional requirements.
- Increases software reliability and robustness.
- Supports early detection and correction of issues.
- Provides a safety net for code changes and refactoring.

Q#5: Types of Abstract Architecture:

- **Layered Architecture**: Organizes components into layers with well-defined responsibilities.
- **Microservices Architecture**: Decomposes the application into small, independently deployable services.
- **Event-Driven Architecture**: Components communicate through events or messages.
- **Service-Oriented Architecture (SOA)**: Systems are composed of services with well-defined interfaces.

Q#6: Two Types of Deployment Models in Cloud Computing:

- **Public Cloud**: Services and infrastructure are owned and operated by third-party providers, accessible to the public.
- **Private Cloud**: Services and infrastructure are used exclusively by a single organization, offering more control and privacy.

Q#7: Reasons to Redesign an Application:

- Changing business requirements.
- Technological obsolescence.
- Poor performance or scalability.
- User feedback and evolving user needs.
- Security vulnerabilities or compliance requirements.
- Enhancing user experience or modernizing the interface.
- Integration with new third-party services or platforms.

1. **What are architecture styles**
2. **What is bad smell in refactoring code**
3. **What are the six skills of software design process**

- **Architecture Styles:**
  - Architecture styles are design patterns or templates that provide a high-level structure for designing software systems. They define the organization, relationships, and interactions among components or modules in a system. Examples include Layered Architecture, Microservices Architecture, Client-Server Architecture, and Event-Driven Architecture.
- **Bad Smell in Refactoring Code:**
  - In refactoring, a "bad smell" refers to a symptom of poorly structured code that indicates a potential problem. Bad smells often suggest that a codebase may need refactoring. Examples of bad smells include duplicated code, long methods or functions, excessive class complexity, and inappropriate naming.
- **Six Skills of Software Design Process:**
  - The six essential skills in the software design process are:
    1. **Problem Decomposition**: The ability to break down complex problems into smaller, manageable parts.
    2. **Abstraction**: The skill to create abstract representations of system components, focusing on key characteristics while hiding unnecessary details.
    3. **Modularity**: Organizing the system into modular components to promote separation of concerns and reusability.
    4. **Pattern Recognition**: Recognizing and applying design patterns and best practices to solve common design problems.
    5. **Algorithmic Thinking**: Developing algorithms and logical solutions to design challenges.
    6. **Trade-off Analysis**: Evaluating trade-offs among design decisions, considering factors like performance, maintainability, and scalability.

Ek question ta aur 5 marks ka usmy table ko match krna ta

1. **Ek ta type of requirements just write there name (5 marks)**
2. **What is software requirements write to the point answer (3 marks)**
3. **Write a short note on stack holder goals list (3 marks)**
4. **Ek ta kuch in abc company you are software architect**
5. **What are the 10 requirements Elicitation technique just write their name. (5 marks)**
6. **Types of Requirements:**
   a. Functional Requirements
   b. Non-Functional Requirements
   c. Business Requirements
   d. User Requirements

    e. System Requirements
7. **Software Requirements:**
    a. Software requirements are descriptions of the functions, capabilities, constraints, and quality attributes that a software system must possess to meet user needs and business goals. They serve as the foundation for software design and development.
8. **Stakeholder Goals List:**
    a. A stakeholder goals list is a document that outlines the objectives and expectations of various stakeholders involved in a project. It helps ensure that the project aligns with the needs and goals of all relevant parties.
9. **Scenario: ABC Company Software Architect:**
    a. In this scenario, you, as a software architect at ABC Company, would likely be responsible for designing the software systems and solutions used within the organization. Your role would involve making architectural decisions, defining system structures, and ensuring that software projects align with the company's goals.
10. **Requirements Elicitation Techniques:**
    a. Interviews
    b. Surveys and Questionnaires
    c. Workshops and Focus Groups
    d. Document Analysis
    e. Observation
    f. Brainstorming
    g. Prototyping
    h. Use Cases and Scenarios
    i. User Stories
    j. Job Shadowing