1. (2pts) True/False --Needs, Features, and Requirements must all be testable. Explain your answer.

*False.Needs and Features do not need to be testable. Requirements must be testable.*

1. (2pts) Explain the difference between a use case flow and a scenario.

*A flow is a set of scenarios with similar goal and outcome. A scenario is one path through a flow.*

1. (4pts) What are 2 general rules to follow when creating a layers architecture? Give an example of each.

*Visibility – layers only depend on the same layer or next lower layer. E.g. UI layer depends on the business logic layer, but not on the data access layer.*

*Volatility – higher layers, more change – lower layers, less change – e.g. application logic in the highest layers – device drivers at the lower layer.*

1. (1pts) True/False--Key abstractions will always be entity classes. Explain your answer.

*False – we could have a boundary class or a control class that we know we will need. E.g. YogaDBAccess boundary class.*

1. (4pts) Explain the difference between an alternate flow and an exceptional flow. Give an example of each.

*An alternate flow explains the system responses needed when a business rule is not met – for example a class is full.*

*An exceptional flow shows the system response when an error condition occurs like a disk access failure, or some other system failure.*

1. (6 pts) List and briefly describe (one phrase for each) the components of a Use Case Description.
   1. *Name – reflects the goal*
   2. *Brief Description – describes the actor and goal*
   3. *Actors – simple list*
   4. *Flow of Events – show user action and system response – can have basic, alternate, and exceptional flows.*
   5. *Pre-Conditions – could be other use cases*
   6. *Post-Conditions – any system changes that persist*
   7. *Business Rules - the business logic for this use case*
2. We update our Yoga Studio Project problem statement to add the following:

***We want to reward our customers who take Yoga classes. Any customer currently enrolled in a Yoga class will receive a 20% discount on our Yoga products.***

* 1. (4pts)Show any modification to the Yoga StudioVision Document needed to address this change.

*There will be either a new Need/Feature pair for this or we could update the purchase product Need/Feature pair.*

* 1. (6pts)What are the components of our Yoga Studio SRS document? For each component specify if that component will change with this new problem statement change.Show anymodifications to the Yoga Studio SRS needed to address this change.

*Use Case Diagram, Use case definitions, and glossary. The only change will be for the*

***Customer Purchase Product*** *use case. It will have this new 20% discount for active students business rule added and there will be a check for this added in the system response for calculate the order price when the user clicks place order.*

1. (6pts)What are the outputs from architecture analysis? Suppose our problem statement changes so that we do not have pre-requisites for any classes. That is***, any customer can take any Yoga class in any sequence***. For each output of architecture analysis indicate if that output will change due to this problem statement change. Explain your reasoning and list the changes needed.

*Layers structure – no change*

*Architectural Mechanisms – no change*

*Key Abstractions – this will change. We no longer need the waivers class. We may not need the faculty members to be advisors, but this need not be reflected in the key abstractions.*

|  |  |  |
| --- | --- | --- |
| * **Essential Difficult to software engineering:** 1) Complexity. 2) Conformity. 3) Changeability. 4) Invisibility. * The Unified Modelling Language (UML) is a language for (SVCD) i.Specifying   ii.Visualizing  iii.Constructing  iv.Documenting   * **Techniques for eliciting requirements:**   a. Interviewing  b. Requirements workshops  c. Brainstorming  d. Storyboarding  e. Use Cases  f. Role playing (become the user for a while)  g. Prototyping   * **Requirements Analysis**   1. Use-case diagram  2. Use-case descriptions   * **Architectural Analysis**   1. Key abstractions  2. Upper level architectural layers   * **Use-case Analysis**   1. Sequence diagrams  2. Collaboration diagrams  3. VOPC diagrams  4. Analysis class to analysis mechanism map   * **Architectural Design**   1. Subsystem context diagram  2. Analysis class to design elements map  3. Design elements to “owning” package map   * **Use-case Design**   1. Updated interaction diagrams including design elements  2. Updated VOPC class diagrams including design elements   * **Subsystem Design**   1. Interaction diagram for each interface definition  2. Class diagram for subsystem   * **Class Design** 1. Refined and updated VOPC class diagrams * **Layers architecture rules to follow:**   • **Visibility**. Subsystems may only depend on subsystems in the same layer and the next lower layer. • **Volatility.** • In the highest layers, put elements which vary when user requirements change. • In the lowest layers, put elements that vary when the implementation platform (hardware, language, operating system, database, etc.) changes. • Sandwiched in the middle, put elements which are generally applicable across wide ranges of systems and implementation environments. • Add layers when an additional partition within these broad categories helps to organize the model. • Generality. Abstract model elements tend to be placed lower in the model.  If not implementation-specific, they tend to gravitate toward the middle layers. | | * **The UML Provides Standardized Diagrams** Use Case Diagrams   Activity Diagrams  Class Diagrams  Object Diagrams  State Diagrams  Sequence Diagrams  Collaboration Diagrams ------------------------ Deployment Diagrams  Component Diagrams   * **Lifecycle Phases: Inception -** *Define the vision and scope of project*   **Elaboration -** *Plan project, specify features, baseline architecture*  **Construction -** *Build product* **Transition -** *Transition product to users*   * **Basic RUP OOAD Activities:** 1.RequirementAnalysis, 2. Architecture Analysis, 3. Use Case Analysis. 4. Architecture Design. 5. Use case Design. 6. Sub System Design. 7. + class design. * **The VOPC** (View of Participating Classes):   **VOPC** Diagram captures the structural relationships among classes.  This is a special kind of class diagram. * **Recommended Steps for Creating Use Cases**   Identify actors  Name use cases  Start with verb, reflect goal  Brief description  Main success scenario  Pre and post conditions  Alternate flows  Exceptional flows—what might go wrong?  Business rules Associated non-functional requirements   * **Supplementary Specification:** Functionality   Usability  Reliability  Performance  Supportability  Design constraints   * 2/3 of life-cycle cost is in system maintenance (for successful projects) * Only about 5% total costs for coding * Fixing faults is much cheaper earlier in life-cycle than later * 60-70% faults in large projects specification / design faults * Greatest Potential (Brooks) Rapid prototyping   Incremental development  Develop great designers   * Best Practices Address Root Causes * An ***ACTOR*** is someone or something outside the   system that interacts with the system   * Use-Cases Include a Flow of Events * Late-phase discovery of design defects results in costly over-runs and/or project cancellation |
| * **RUP Key Features:** Use Case Driven. Iterative and incremental (Models, workflows, phases, and iterations) Architecture centric | | * Identifying Analysis Mechanisms approaches: ***• Top Down*** -- use well-known approaches in the context of your new application (e.g. persistence) • ***Bottom Up*** -- identify some pattern in emerging application that is relevant for the current application and possibly others -- e.g., a rules engine or other useful framework |
| * **Benefits of a Use-Case Driven Process** Use-Cases are concise, simple, and understandable by a wide range of stakeholders Use-Cases help synchronize the content of different models | | * Traditional Waterfall Development -> Requirement Analysis, -> Design, -> Code & Unit Testing, -> Subsystem Testing, -> System Test * Waterfall Development Delays Reduction of Risk * Iterative Development Accelerates Risk Reduction |
| * **Benefits of Iterative Development** Serious misunderstandings become evident early in the life cycle   Enables and encourages user feedback  Development focuses on critical issues  Objective assessment thru testing  Inconsistencies detected early  Workload of teams is spread out  Leverage lessons learned earlier  Stakeholders are kept up to date on project’s status | | * Benefits of using Component Architectures Components facilitate resilient architectures   Modularity enables a clear separation of concerns among elements of a system that are subject to change.  Reuse is facilitated by leveraging standardized frameworks and commercially available components   * A ***STAKEHOLDER*** is anyone who represents an interest group whose needs must be satisfied by the project. * A ***USE CASE*** is a sequence of actions performed by an actor interacting with the system to achieve a goal |
| * **Sample Analysis Mechanisms Distribution of componen**ts on multiple tiers or via web services – e.g. SOAP or RESTful   Transaction management  Concurrency  Persistence -- e.g. Object Relational Mapping - ORM  Security -- e.g. Java Authentication and Authorization  Error and Exception detection / handling / reporting  Rules engine  Wrapping of legacy systems  Web Application Framework  User Interface Framework   * Responsibilities from Collaboration Diagram Responsibilities are specifications of object behaviour * **Attributes** a named property of a class that describes a range of values that instances of the property may hold. * **Sequence vs Collaboration Diagrams** | | * **Diagram Scenarios vs Flows vs Use Case?** A scenario is a single pass   Straightforward to create and read  A flow is a set of similar scenarios  Might involve conditions or loops  Separate diagrams for significant flows  A use-case is the set of all flows  In general, too complex for interaction diagrams  Full coverage infeasible  Sequence diagrams are most useful to show what classes we will need and their interactions  Sequence diagram are not very good at modeling complex logic flow   * **Our Sequence Diagram Conventions** Label our analysis classes with <<Boundary>>, <<Control>>, or <<Entity>>. We show communication with our Data Base thru a data base boundary class. ORM is part of our OO Analysis and Design. * A ***SCENARIO*** is a single path through a use case. * A ***FLOW*** is a set of scenarios that result in the same sort of outcome |
| Sequence Diagrams  Show the explicit sequence of messages  Better for visualizing overall flow  Better for real-time  Specifications | Collaboration Diagrams  Show relationships in addition to interactions (use for VOPC diagrams)  Better for visualizing patterns of collaboration  Better for visualizing all of the effects on a given project |  |