1. (T/F) Need, Features and Requirements must be testable, Explain

Ans:

**False**.

Requirements are testable. The Needs and Features discussed in the Vision document are further refined into a list of detailed requirements. Requirements differ from needs and features.

1. Explain the difference between a use case flow and scenario.

Ans:

A use case “flow” is a collection of scenarios that have a common structure. Like several boats floating together on the current. A use case has one normal, basic flow and possibly several other alternative flows to handle: Regular variants (business logic alternatives) Odd cases

**Scenario**: A scenario is an instance of a use case like one boat on the river.

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

Ans:

1. True/False – key abstraction will always be entity classes. Explain your answer.
2. Explain the difference between an alternate flow and an exceptional flow. Give an example of each.

Ans:

Result negative: An Exception is anything that leads to NOT achieving the use case’s goal.

Result positive: An Alternate Flow is a step or a sequence of steps that achieves the use case’s goal following different steps than described in the main success scenario. But the goal is achieved finally.

An alternate flow describes a scenario other than the basic flow that results in a user completing his or her goal. It is often considered to be an optional flow.

Exceptional flows handle error situations (like system, DB errors, etc.) Things we might catch with exceptions of some type.

1. List the components of a use case description

**Lesson 2**

1. Essential difficulties/challenges in software development.
   1. Complexity: complexity of process and state, sheer size, distributed, network
   2. Conformity: hardware compatibility
   3. Changeability: requirement for modification
   4. Invisibility: software is abstract
2. Importance of analysis and design

Fixing faults in earlier life cycle is much cheaper than later

Good Analysis & Design facilitates integration and maintenance

*Two-thirds of all the faults in large scale projects have been observed to be specification or design faults*

**What is UML and why use UML?**

UML is a standard modeling language, helps to ***maintain consistency*** among *system’s artifacts* and *facilitates communication among team members*.

The Unified Modeling Language (UML) is a language for

Specifying, Visualizing, Constructing, Documenting

**What Is a Software Development Process?**

SDP Is a ***set of activities*** needed to transform a user’s requirements into a software system. The process defines who is doing What, When, as well as How to reach a certain goal.

Key Features of RUP (Rational Unified Process)

* Use case Driven
* Iterative and incremental
* Architecture Centric

**What is Actor?**

An actor is someone or something outside the system that interacts with the system.

**What is use-case?**

A use-case is a sequence of actions performed by a system to produce a result for a particular actor.

**What is the benefits of a Use-Case Driven Process?**

***Use-Cases*** are concise, simple, and understandable by a wide range of stakeholders. End users, developers and acquirers understand *functional requirements* of the system

***Use-Cases*** drive numerous activities in the process.

***Use-Cases*** help *synchronize* the content of *different models*.

**Lifecycle Phases of RUP:**

The Rational Unified Process (RUP) has four phases:

1. *Inception* - Define the vision and scope of project
2. *Elaboration* - Plan project, specify features, baseline architecture
3. *Construction* - Build product
4. *Transition* - Transition product to users

**Benefits of Iterative Development?**

* Development focuses on critical issues
* Inconsistencies detected early
* Workload of teams is spread out
* Stakeholders are kept up to date on project’s status

**Lesson 3 (System Analysis and Requirements)**

**Who Are The Stakeholders?**

A Stakeholder is anyone who represents a group whose needs must be satisfied by the project.

Selecting stakeholders for a project requires careful consideration for each project.

*Typical examples:*

Customer or customer representative, • User or user representative, • Investor, • Shareholder, • Owner, • Board member, • Production manager, • Buyer, • Designer, • Tester, • Documentation writer

**The goal of System Analysis:** *The goal of System Analysis is ultimately a* ***Software Requirements Specification. (Is to provide software requirements specification)***

**Problems → Needs → Features**

Problems are reformulated as user needs, and needs are reformulated as features for the new system.

The difference between Problems, Needs, and Features is a matter of both detail and orientation.

Every *problem* should be ***mapped*** to *one or more needs*, and *every need* should be **mapped** to *one or more features* that would meet the need.

Example:

|  |  |  |
| --- | --- | --- |
| **Problem** | **Need** | **Feature** |
| Paper catalogs are clumsy and  unsatisfactory to customers | Online catalog | a) System should support online browsing of catalog  b) System should support online purchases |

**The Vision Document**

The result of Problem Analysis is Vision Document. It provides the following things:

It documents

The problem

The key and features

The business case for the project and clarifies the scope of the project.

**Provide** list of stakeholders. The contractual basis for the more detailed technical requirements.

***From RUP perspective***: Core requirements of stakeholders are specified as key needs and Needs

**Features → Requirements**

Needs and features of vision documents are refined to a list of requirements.

**Techniques for eliciting/creating requirements:**

* Interviewing
* Requirements workshops
* Brainstorming
* Storyboarding
* Use Cases
* Role playing (become the user for a while)
* Prototyping

**Lesson 4**

**What is Requirements Analysis? What different types of role of Requirement analysis?**

Requirement analysis is an approach to capture functional system requirements.

*Roles*:

**System Analyst role:** coordinates requirements creation and use-case modeling

**Requirements Specifier role:** specifies the details of one or more parts of the system's functionality.

**Architectural Analysis:**

It provides a *comprehensive view of the* *problem domain* and *high level system design* that will be used across all use-case realizations.

*Role*: The software architect role is responsible for the software architecture, which includes the key technical decisions that constrain the overall design and implementation for the project.

**Use Case Analysis:**

Analysis of business domain and user requirements. Use-case includes

1. Sequence diagrams shows how participating classes carry out each use-case

2. Collaboration diagrams reflect required **associations** among classes.

3. The VOPC diagram captures the **structural relationships** among classes. This is a special kind of class diagram

*Role:*

The designer role is responsible for designing a part of the system, within the constraints of the requirements, architecture, and development process for the project.

**Architectural Design:**

The architect coordinates the design activities of individual designers.

* Identify design classes
* Identify packages
* Identify subsystems and their interfaces
* Identify the system layering strategy
* Identify reusable parts at the system level
* Identify components/tools and technology used for the whole project.

**Use Case Design**

During use case design designers *inspect and elaborate architectural design* using the use cases.

**Subsystem Design**

Subsystem design basically follows the same steps like analysis and use case design for single sub system. As in use case design, main outputs are **interaction** (dynamic) and **class** (static) **diagrams** of the subsystem elements.

**Class Design**

Class design is a final preparation for implementation. Class design is making final design refinements. Class design *focus on* *refining the operations, attributes, and the relationships* for all our classes.

**Lesson 5 (The Use Case Approach to Software Requirements)**

**1**. The main elements of a **RUP Software Requirements Specification**:

- Use case models, - supplementary specifications, - glossary.

**2**. A **use case** is a sequence of actions performed by an actor interacting with the system to achieve a goal, showing how the goal might succeed or fail to be reached.

**3**. Use cases are **described** in terms of flows and scenarios. 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 (e.g., success vs failure flows).

* A use case is a sequence of actions performed by an actor interacting with the system to achieve a goal. Goal and the use case can often be used interchangeably
* Use cases are described in terms of flows and scenarios. 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 (e.g., success vs failure flows).

**How Use Cases Drive Overall Development?**

Use cases form the basis of a modern SRS document.

* *Organizing* the requirements document using user goals.
* *Actions* facilitates user understanding of the requirements.

*The use-case model is composed of: 1. use-case diagram(s) 2. use-case descriptions*

*Actors are External to the System*

**What are Use Case Descriptions?**

* Name
* Brief description
* Flows of Events
* Relationships
* Activity and State
* Diagrams
* Use-Case diagrams
* Special requirements
* Preconditions
* Post conditions
* Other diagrams

**What Are Scenarios?**

A scenario is an instance of a use case. One path. Like one boat on the river

**Use-Case Flows of Events**

A use case “flow” is a collection of scenarios that have a common structure

* A use case has **one normal basic flow** and possibly several other alternative flows to handle: Regular variants (business logic alternatives) and Odd cases
* Exceptional flows handle error situations (like system, DBerrors, etc.)
* Note that **alternate flows** are used to explain the system response to **Business Rule checks**:

**Use Cases should have these:**

1. Should have a name proper name to suggest a goal of the use case.
2. Should include a set of successful path from a trigger event to the goal
3. Should include a set of paths from a trigger event to the failure.

**Use Cases DON'Ts**

* A Use Case should not specify user **interface design.**
* A Use Case should not specify **implementation detail** that may have been imposed on the project.

**Use Case Relationships: Include**

* Subgoals/subflows
* Login, print, valuation

**Supplementary Specs (Nonfunctional Requirements)**

* Reliability, performance, security, usability, Supportability

*Reliability*

The main system must be running 95% of the time.

*Performance*

The system shall support up to 15 simultaneous users against the central database at any given time.

*Design Constraints*

The system shall integrate with an existing legacy system

**Glossary**

Facilitates common understanding among developers as well as domain experts

Example:

Library Member: Person who checks-out books from the library.

Librarian: Person who works for the library and performs the check-in and check-out Procedure.

Check-out Book: The procedure done if a library member returns a checked out book to the library.

**Design pattern**

* A design pattern is a customizable solution to a common design problem.
* Architectural patterns are design patterns applied to software architecture. Example: the Layering Pattern

Architectural Layers Rules:

Visibility: subsystem may only depends on the subsystem in the same layer and the next lower layer.

Volatility: elements those are changeable should be in highest level. And elements those do not change frequently should be lower level. Elements generally applicable wide ranges of system are put in the middle layer.

**Architectural mechanisms:** At the analysis stage the design pattern such as layers, MVC etc are called architectural mechanism.

**Architectural Mechanisms**

* Provides reusability, ease of maintenance and maximizes entropy of multiple solution strategies.
* An architectural mechanism is really an architectural pattern for the project.
* **Whenever possible, represent in diagrams as a placeholder**

Example: Plan to use a legacy database (since it is a known constraint) and represent the database as a placeholder in diagrams.

**Analysis Mechanisms**

Two approaches: Top Down (persistence), Bottom up (a rules engine)

* Key abstractions provide a common set of **key domain elements** across use case realizations.

**Use-Case Analysis Steps**