Q1 Answer the following questions: (5x5=25)

(a) What are primary and secondary actors? Give examples.
(b) What is the use of stubs and Drivers in testing?
(c) Discuss the types of relationships and associations in class diagrams.
(d) What is the difference between Sequence diagrams and Collaboration diagrams?
(e) Discuss the benefits of OOSE over traditional SE.

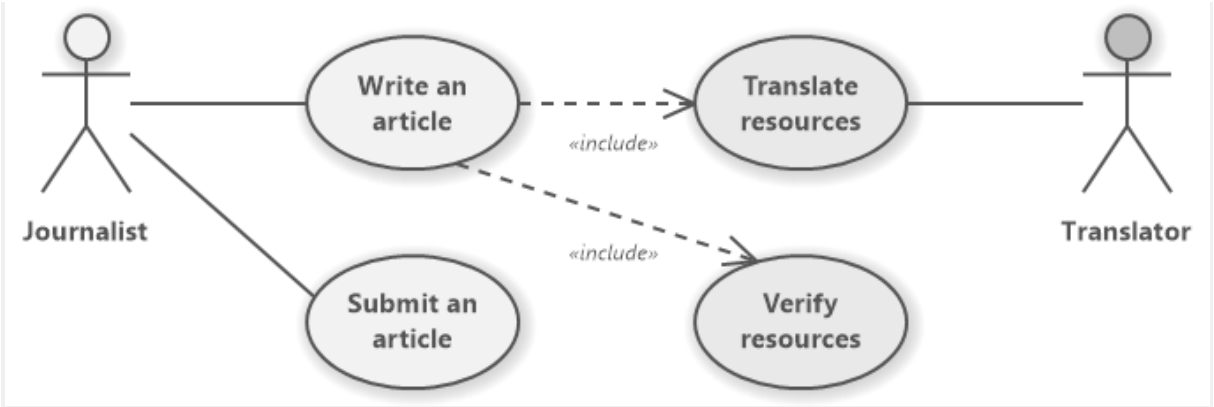
The **primary actor** of a use case is the actor which specifies the user goal and usually also initiates an interaction with the use case.

The **secondary actor** (also called supporting actor) never initiates an interaction with the use case, and it does not trigger it in any way. The secondary actor is involved in the use case to provide a service for the system as the reaction to the use case actions.

The primary actors are usually placed on the left side of the diagram. The secondary actors are placed on the right side by the convention. (The placement is not required by the standard, only recommended for easier diagram reading.)

Primary and Secondary Actor Example

Our use case diagram example shows two actors within the editorial office system - Journalist and Translator. The Journalist is a primary actor - they can write and submit articles. The Translator is a secondary actor in our system. The actor is called by the Journalist - within the Translate resources use case. The Translator never triggers the Translate resources use case.

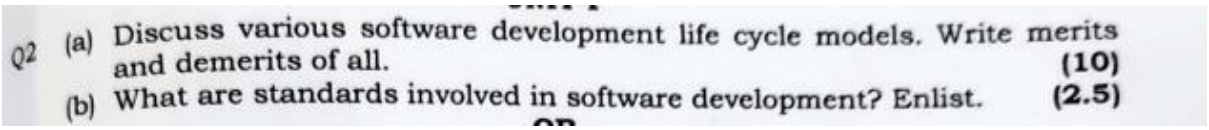


S.No.	Stubs	Drivers
1.	Stubs are used in Top-Down Integration Testing.	Drivers are used in Bottom-Up Integration Testing.
2.	Stubs are basically known as a “called programs” and are used in the Top-down integration testing.	While, drivers are the “calling program” and are used in bottom-up integration testing.
3.	Stubs are similar to the modules of the software, that are under development process.	While drivers are used to invoking the component that needs to be tested.
4.	Stubs are basically used in the unavailability of low-level modules.	While drivers are mainly used in place of high-level modules and in some situation as well as for low-level modules.
5.	Stubs are taken into use to test the feature and functionality of the modules.	Whereas the drivers are used if the main module of the software isn’t developed for testing.
6.	The stubs are taken into concern if testing of upper-levels of the modules are done and the lower-levels of the modules are under developing process.	The drivers are taken into concern if testing of lower-levels of the modules are done and the upper-levels of the modules are under developing process.
7.	Stubs are used when lower-level of modules are missing or in a partially developed phase, and we want to test the main module.	Drivers are used when higher-level of modules are missing or in a partially developed phase, and we want to test the lower(sub)- module.

Sequence Diagrams	Collaboration Diagrams
The sequence diagram represents the UML, which is used to visualize the sequence of calls in a system that is used to perform a specific functionality.	The collaboration diagram also comes under the UML representation which is used to visualize the organization of the objects and their interaction.
The sequence diagram are used to represent the sequence of messages that are flowing from one object to another.	The collaboration diagram are used to represent the structural organization of the system and the messages that are sent and received.
The sequence diagram is used when time sequence is main focus.	The collaboration diagram is used when object organization is main focus.
The sequence diagrams are better suited of analysis activities.	The collaboration diagrams are better suited for depicting simpler interactions of the smaller number of objects.

Benefits of OOSE

- Modular, scalable, extensible, reusable, and maintainable.
- It models the complex problem in a simple structure.
- Object can be used across the program.
- Code can be reused.
- We can easily modify, append code without affecting the other code blocs.
- Provides security through encapsulation and data hiding features.
- Beneficial to collaborative development in which a large project is divided into groups.
- Debugging is easy.


Q2 (a) Discuss various software development life cycle models. Write merits and demerits of all. (10)
(b) What are standards involved in software development? Enlist. (2.5)

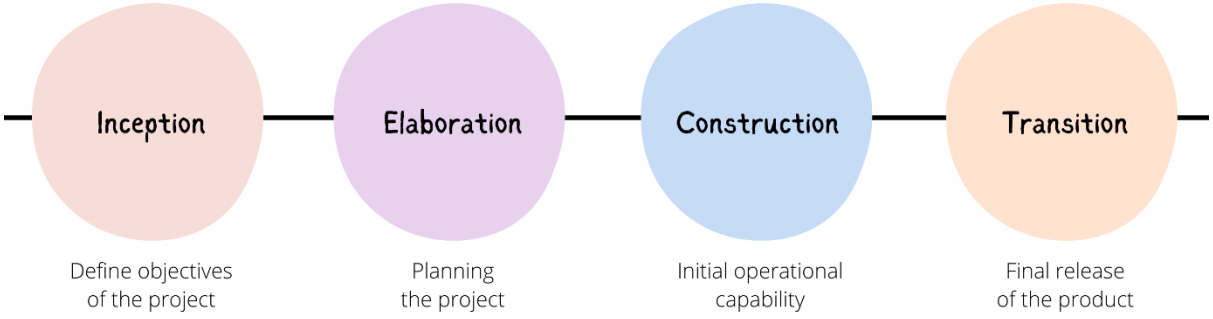
https://www.javatpoint.com/software-engineering-sdlc-models

(a) What are the phases of Object Oriented Unified processing? Explain. **(4)**
(b) Differentiate between verification and validation. **(3)**
(c) What is the use of Requirement Elicitation? Explain at least two techniques used for requirement elicitation. **(5.5)**

Phases

We can represent a unified process model as a series of cycles. Each cycle ends with the release of a new system version for the customers. We have four phases in every cycle:

- Inception
- Elaboration
- Construction
- Transition



Verification	Validation
It includes checking documents, design, codes and programs.	It includes testing and validating the actual product.
Verification is the static testing.	Validation is the dynamic testing.
It does *not* include the execution of the code.	It includes the execution of the code.
Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.

Verification	Validation
It checks whether the software conforms to specifications or not.	It checks whether the software meets the requirements and expectations of a customer or not.
It can find the bugs in the early stage of the development.	It can only find the bugs that could not be found by the verification process.
The goal of verification is application and software architecture and specification.	The goal of validation is an actual product.
Quality assurance team does verification.	Validation is executed on software code with the help of testing team.
It comes before validation.	It comes after verification.
It consists of checking of documents/files and is performed by human.	It consists of execution of program and is performed by computer.

Advantages of Requirements Elicitation:

- Helps to clarify and refine the customer requirements.
- Improves communication and collaboration between stakeholders.
- Increases the chances of developing a software system that meets the customer needs.
- Avoids misunderstandings and helps to manage expectations.
- Supports the identification of potential risks and problems early in the development cycle.
- Facilitates the development of a comprehensive and accurate project plan.
- Increases user and stakeholder confidence in the software development process.
- Supports the identification of new business opportunities and revenue streams.

Requirement elicitation Methods:
There are a number of requirements elicitation methods. Few of them are listed below –

1. Interviews
2. Brainstorming Sessions
3. Facilitated Application Specification Technique (FAST)
4. Quality Function Deployment (QFD)
5. Use Case Approach

The success of an elicitation technique used depends on the maturity of the analyst, developers, users, and the customer involved.

1. Interviews:
Objective of conducting an interview is to understand the customer's expectations from the software.
It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility.
Interviews maybe be open-ended or structured.

1. In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
2. In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views
- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

3. Facilitated Application Specification Technique:
It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.
A team oriented approach is developed for requirements gathering.
Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4. Quality Function Deployment:
In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.
3 types of requirements are identified –

- **Normal requirements –**
  In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc
- **Expected requirements –**
  These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
- **Exciting requirements –**
  It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

The major steps involved in this procedure are –

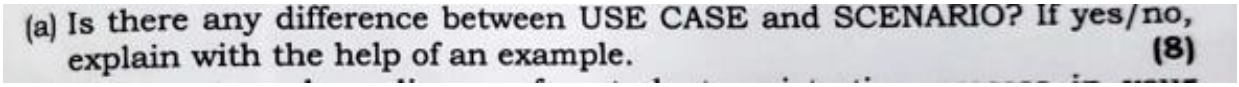
1. Identify all the stakeholders, eg. Users, developers, customers etc
2. List out all requirements from customer.
3. A value indicating degree of importance is assigned to each requirement.
4. In the end the final list of requirements is categorized as –
   - It is possible to achieve
   - It should be deferred and the reason for it
   - It is impossible to achieve and should be dropped off

5. Use Case Approach:
This technique combines text and pictures to provide a better understanding of the requirements.
The use cases describe the 'what', of a system and not 'how'. Hence, they only give a functional view of the system.
The components of the use case design includes three major things – Actor, Use cases, use case diagram.

1. **Actor –**
   It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
   - Primary actors – It requires assistance from the system to achieve a goal.
   - Secondary actor – It is an actor from which the system needs assistance.
2. **Use cases –**
   They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the

system. A complete set of use cases specifies all possible ways to use the system.

3. **Use case diagram –**
   A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
   - A stick figure is used to represent an actor.
   - An oval is used to represent a use case.
   - A line is used to represent a relationship between an actor and a use case.


(a) Is there any difference between USE CASE and SCENARIO? If yes/no, explain with the help of an example. (8)

1. A scenario is a situation in which more than one actor is involved in performing a specific task. A use case is the description of the scenario which involves describing the functionality of the scenario.
2. The scenario is the instance of the use case whereas the use case is the external entity such as an actor or system.
3. The scenario doesn't contain entry or exit points but the use case has this for describing the conditions within the range.

Example-

1. Scenario-

Consider a university that requires an application for an online exam. The scenario describes the use of an online exam, an individual such as a student, and the equipment such as a system.

The scenario for this application is-

- Authentication of the student
- The URL for the student which is an exam URL
- List of options in the exam for the student.
- List of questions for the students.
- Log-off information of the student.

2. Use cases-

Consider the online exam system. The use case is the online exam. The actor is a student and the system is server.

The flow of the events-

- The student connects to the server
- The student selects the exam.
- The student submits the exam
- The student log off the system.

Entry condition-

- The system verifies the student login information

Exit condition-

- Successful log off the system

(a) What are the different types of objects in Analysis model?

- **Entity objects** -- these represent persistent information tracked by a system. This is the closest parallel to "real world" objects.
- **Boundary objects** -- these represent interactions between user and system. (For instance, a button, a form, a display)
- **Control objects** -- usually set up to manage a given usage of the system. Often represent the control of some activity performed by a system

Q8 Explain **any two** of the following:-
(a) Integration Testing

Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group.

Types of Integration Testing

- Big Bang Approach:
- Incremental Approach: is further divided into the following
  - Top-Down Approach
  - Bottom-Up Approach
  - Sandwich Approach – Combination of Top Down and Bottom Up

Big Bang Testing is an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit. This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.

Advantages:

- Convenient for small systems.

Disadvantages:

- Fault Localization is difficult.

Incremental Testing

In the **Incremental Testing** approach, testing is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application. Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully.
Incremental Approach, in turn, is carried out by two different Methods:

- Bottom Up
- Top Down

Stubs and Drivers

Stubs and Drivers are the dummy programs in Integration testing used to facilitate the software testing activity. These programs act as a substitutes for the missing models in the testing. They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing.
Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

Bottom-up Integration Testing

Bottom-up Integration Testing is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower-level modules are tested and integrated, then the next level of modules are formed.
Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible

Top-down Integration Testing

Top Down Integration Testing is a method in which integration testing takes place from top to bottom following the control flow of software system. The higher level modules

are tested first and then lower level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.

Advantages:

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

Sandwich Testing

Sandwich Testing is a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system. It is a combination of Top-down and Bottom-up approaches therefore it is called **Hybrid Integration Testing**. It makes use of both stubs as well as drivers.

(b) State Based Testing

Objectives of State Transition Testing:

The objective of State Transition testing is:

- To test the behavior of the system under varying input.
- To test the dependency on the values in the past.
- To test the change in transition state of the application.
- To test the performance of the system.

Transition States:

- **Change Mode:**
  When this mode is activated then the display mode moves from TIME to DATE.
- **Reset:**
  When the display mode is TIME or DATE, then reset mode sets them to ALTER TIME or ALTER DATE respectively.
- **Time Set:**
  When this mode is activated, display mode changes from ALTER TIME to TIME.
- **Date Set:**
  When this mode is activated, display mode changes from ALTER DATE to DATE.

(c) Testing Process

- **Step-1: Assess Development Plan and Status –**
  This initiative may be prerequisite to putting together Verification, Validation, and Testing Plan want to evaluate implemented software solution. During this step, testers challenge completeness and correctness of event plan. Based on extensiveness and completeness of Project Plan testers can estimate quantity of resources they're going to get to test implemented software solution.

- **Step-2: Develop the Test Plan –**
  Forming plan for testing will follow an equivalent pattern as any software planning process. The structure of all plans should be an equivalent, but content will vary supported degree of risk testers perceive as related to software being developed.

- **Step-3: Test Software Requirements –**
  Incomplete, inaccurate, or inconsistent requirements cause most software failures. The inability to get requirement right during requirements gathering phase can also increase cost of implementation significantly. Testers, through verification, must determine that requirements are accurate, complete, and they do not conflict with another.

- **Step-4: Test Software Design –**
  This step tests both external and internal design primarily through verification techniques. The testers are concerned that planning will achieve objectives of wants, also because design being effective and efficient on designated hardware.

- **Step-5: Build Phase Testing –**
  The method chosen to build software from internal design document will determine type and extensiveness of testers needed. As the construction becomes more automated, less testing are going to be required during this phase. However, if software is made using waterfall process, it's subject to error and will be verified. Experience has shown that it's significantly cheaper to spot defects during development phase, than through dynamic testing during test execution step.

- **Step-6: Execute and Record Result –**
  This involves testing of code during dynamic state. The approach, methods, and tools laid out in test plan are going to be wont to validate that executable code actually meets stated software requirements, and therefore the structural specifications of design.

- **Step-7: Acceptance Test –**
  Acceptance testing enables users to gauge applicability and usefulness of software in performing their day-to-day job functions. This tests what user believes software should perform, as against what documented requirements state software should perform.

- **Step-8: Report Test Results –**
  Test reporting is continuous process. It may be both oral and written. It is important that defects and concerns be reported to the appropriate parties as early as possible, so that corrections can be made at the lowest possible cost.

- **Step-9: The Software Installation –**
  Once test team has confirmed that software is prepared for production use, power to execute that software during production environment should be tested. This tests interface to operating software, related software, and operating procedures.

- **Step-10: Test Software Changes –**
  While this is often shown as Step 10, within context of performing maintenance after software is implemented, concept is additionally applicable to changes throughout implementation process. Whenever requirements change, test plan must change, and impact of that change on software systems must be tested and evaluate.

- **Step-11: Evaluate Test Effectiveness –**
  Testing improvement can best be achieved by evaluating effectiveness of testing at top of every software test assignment. While this assessment is primarily performed by testers, it should involve developers, users of software, and quality assurance professionals if function exists within the IT organization.