Unit – V

Testing

Object Oriented Methodologies –Software Quality Assurance – Impact of Object Orientation on Testing – Develop Test cases and Test Plans.

OBJECT ORIENTED METHODOLOGIES P-1

What is Object Oriented Methodology?

It is a new system development approach, encouraging and facilitating re-use of software components.

It employs international standard Unified Modeling Language (UML) from the Object Management Group (OMG).

Using this methodology, a system can be developed on a component basis, which enables the effective re-use of existing components, it facilitates the sharing of its other system components.

Object Oriented Methodology asks the analyst to determine

- ✓ What the objects of the system are?
- ✓ What responsibilities and relationships an object has to do with the other objects?
- ✓ How they behave over time?

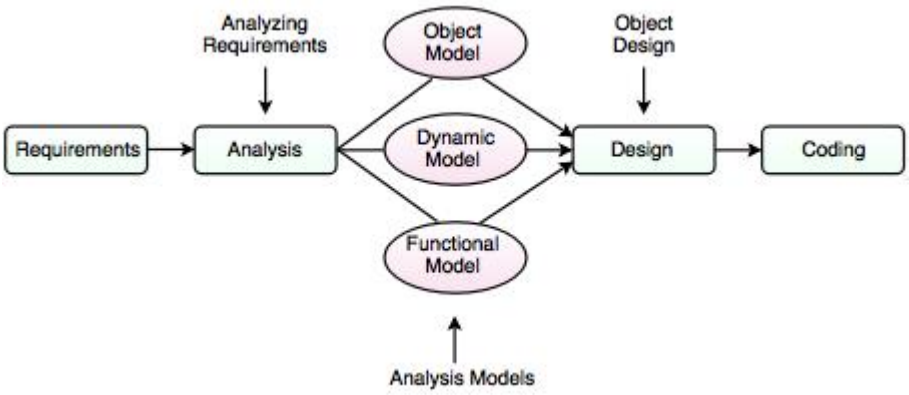
There are three types of Object Oriented Methodologies

1. Object Modeling Techniques (OMT)
2. Object Process Methodology (OPM)
3. Rational Unified Process (RUP)

1. Object Modeling Techniques (OMT)

It was one of the first object oriented methodologies and was introduced by Rumbaugh in 1991.

OMT uses three different models that are combined in a way that is analogous to the older structured methodologies.



a. Analysis

The main goal of the analysis is to build models of the world.

The requirements of the users, developers and managers provide the information needed to develop the initial problem statement.

b. OMT Models

I. Object Model

It depicts the object classes and their relationships as a class diagram, which represents the static structure of the system.

It observes all the objects as static and does not pay any attention to their dynamic nature.

II. Dynamic Model

It captures the behavior of the system over time and the flow control and events in the Event-Trace Diagrams and State Transition Diagrams.

It portrays the changes occurring in the states of various objects with the events that might occur in the system.

III. Functional Model

It describes the data transformations of the system.

It describes the flow of data and the changes that occur to the data throughout the system.

c. Design

It specifies all of the details needed to describe how the system will be implemented.

In this phase, the details of the system analysis and system design are implemented.

The objects identified in the system design phase are designed.

2. Object Process Methodology (OPM)

It is also called as second generation methodology.

It was first introduced in 1995.

It has only one diagram that is the Object Process Diagram (OPD) which is used for modeling the structure, function and behavior of the system.

It has a strong emphasis on modeling but has a weaker emphasis on process.

It consists of three main processes:

I. Initiating: It determines high level requirements, the scope of the system and the resources that will be required.

II. Developing: It involves the detailed analysis, design and implementation of the system.

III. Deploying: It introduces the system to the user and subsequent maintenance of the system.

3. Rational Unified Process (RUP)

It was developed in Rational Corporation in 1998.

It consists of four phases which can be broken down into iterations.

- ✓ Inception
- ✓ Elaboration
- ✓ Construction
- ✓ Transition

Each iteration consists of nine work areas called disciplines.

A discipline depends on the phase in which the iteration is taking place.

For each discipline, RUP defines a set of artefacts (work products), activities (work undertaken on the artefacts) and roles (the responsibilities of the members of the development team).

Objectives of Object Oriented Methodologies

- To encourage greater re-use.
- To produce a more detailed specification of system constraints.
- To have fewer problems with validation (Are we building the right product?).

Benefits of Object Oriented Methodologies

1. It represents the problem domain, because it is easier to produce and understand designs.

2. It allows changes more easily.

3. It provides nice structures for thinking, abstracting and leads to modular design.

4. Simplicity:

The software object's model complexity is reduced and the program structure is very clear.

5. Reusability:

It is a desired goal of all development process.

It contains both data and functions which act on data.

It makes easy to reuse the code in a new system.

Messages provide a predefined interface to an object's data and functionality.

6. Increased Quality:

This feature increases in quality is largely a by-product of this program reuse.

7. Maintainable:

The OOP method makes code more maintainable.

The objects can be maintained separately, making locating and fixing problems easier.

8. Scalable:

The object oriented applications are more scalable than structured approach.

It makes easy to replace the old and aging code with faster algorithms and newer technology.

9. Modularity:

The OOD systems are easier to modify.

It can be altered in fundamental ways without ever breaking up since changes are neatly encapsulated.

10. Modifiability:

It is easy to make minor changes in the data representation or the procedures in an object oriented program.

11. Client/Server Architecture:

It involves the transmission of messages back and forth over a network.

Rumbaugh's Object Modeling Technique: P-2

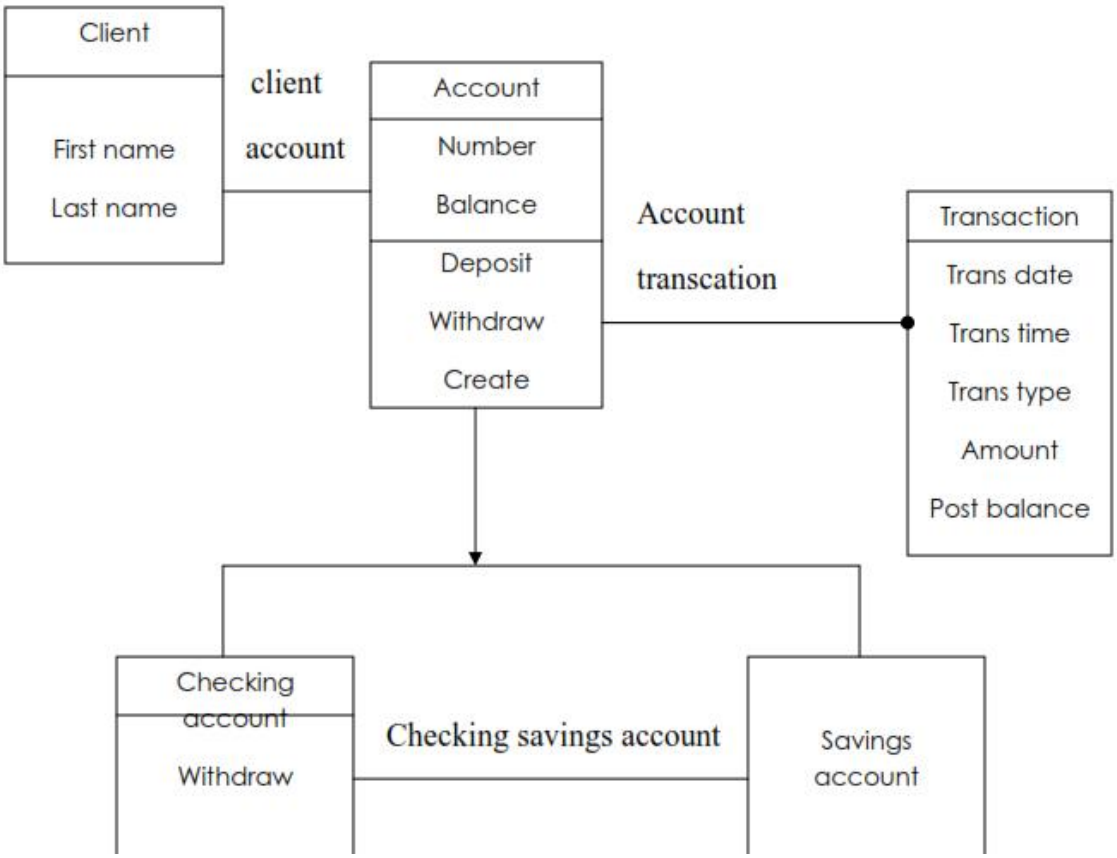
• Describes the dynamic behavior of objects in a system using the OMT dynamic model.
Four phases

Analysis – results are objects, dynamic and functional models.
System design – gives a structure of the basic architecture.
Object design – produces a design document.
Implementation – produces reusable code.

• OMT separates modeling in to three different parts Object Model – presented by object model and the data dictionary.

- Dynamic model - presented by the state diagrams and event
- Flow diagrams. Functional Model – presented by data flow and constraints.
- OBJECT MODEL:-¬ o Object model describes the structure of objects in a system, their identity and relationships to other objects, attributes, and operations. o The object model is represented graphically with an object diagram.

The Rumbaugh method is listed first because it is these authors favorite, and we find it a very friendly and easy methodology. For traditional system analyst's, the Rumbaugh's methodology is the closest to the traditional approach to system analysis and design, and beginners will recognize familiar symbols and techniques. The Rumbaugh methodology has its primary strength in object analysis but it also does an excellent job with object design. Rumbaugh has three deliverables to the object analysis phase; the Object model, the Dynamic model, and the functional model. These three models are similar to traditional system analysis, with the additions for the object model, including definitions of classes along with the classes variables and behaviors. The Rumbaugh object model is very much like an entity relationship diagram except that there are now behaviors in the diagram and class hierarchies. The dynamic model is a "state transition" diagram that shows how an entity changes from one state to another state. The functional model is the equivalent of the familiar data flow diagrams from a traditional systems analysis.



The Booch method P-3

Booch's methodology has its primary strength in the object system design. Grady Booch has included in his methodology a requirements analysis that is similar to a traditional requirements analysis, as well as a domain analysis phase. Booch's object system design method has four parts, the logical structure design where the class hierarchies are defined, the physical structure diagram where the object methods are described. In addition, Booch defines the dynamics of classes in a fashion very similar to the Rumbaugh method, as well as an analysis of the dynamics of object instances, where he describes how an object may change state.

Booch methodology is a widely used object-oriented method that helps to design your system using the object paradigm.

• Is criticized for his large set of symbols.

•It consists of the following diagrams:

- Class diagrams.
- Object diagrams.

- State transition diagrams.
- Module diagrams.
- Process diagrams.
- Interaction diagrams.

• Two processes:

- Macro development process
- Micro development process.

Macro development process.

Primary concern – technical management of the system.

Steps involved:

Conceptualization.

Establish the core requirements and develop a prototype.

Analysis and development of the model

Use the class diagram to describe the roles and responsibilities of objects. Use the object diagram to describe the desired behavior of the system.

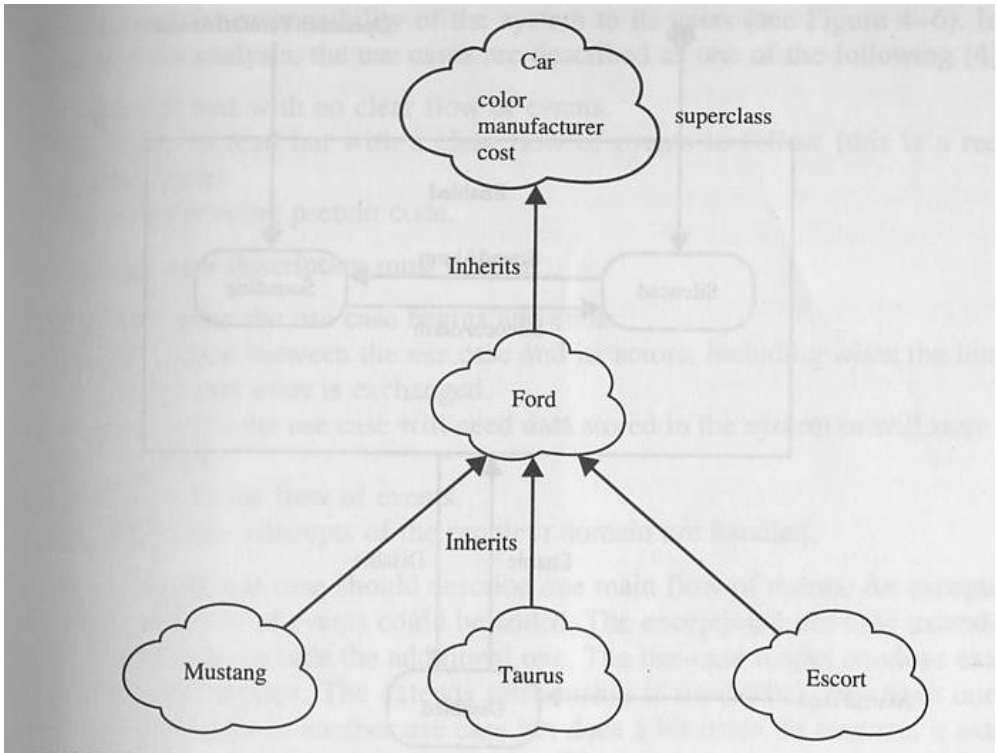


FIGURE 4–4
Object modeling using Booch notation. The arrows represent specialization; for example, the class Taurus is subclass of the class Ford.

JACOBSON METHODOLOGY:

• Object-oriented business engineering (OOBE)
• Object-oriented software engineering (OOSE)
  – It covers the entire life cycle
  – Stress traceability(enables reuse of analysis and design work) both forward and backward
• Use cases
  - Scenarios for understanding system requirements.
  - Non formal text with no clear flow of events.
  - Text easy to read.
  - Formal style using pseudo code.
  - Can be viewed as concrete or abstract (not initiated by actors).

– Understanding system requirements
– Interaction between user and system
– It captures the goal of the user and responsibility of the system to its users

Object oriented software Engineering: Objectory

• OOSE is also called Objectory
• Development process is also called as use case driven development
• The system development method based on OOSE is a process for the industrialized development of the s/w

Use case model:-

The use-case model defines the outside and inside of the system behaviour

Domain Object Model:-

The objects of the real worls are mapped in to the domain object model

Analysis Object Model:-

The analysis object model presents how the source code should be carried out written

Implementation model:-

The implementation model represents the implementation of the system

Test model:-

The test model constitutes the test plans, specification and reports.

• Object oriented business Engineering

Analysis phase:-

The analysis phase defines the system to be built in terms of the
problem-domain object model
the requirements model
analysis model

Design and Implementation phase:-

o The implementation environment must be identified for the design model.

Testing phase:-

o This level includes unit testing, integration testing and system testing.

Software Quality

Schulmeyer and McManus have defined software quality as “the fitness for use of the total software product”. A good quality software does exactly what it is supposed to do and is interpreted in terms of satisfaction of the requirement specification laid down by the user.

Quality Assurance

Software quality assurance is a methodology that determines the extent to which a software product is fit for use. The activities that are included for determining software quality are −

- Auditing
- Development of standards and guidelines
- Production of reports
- Review of quality system

Quality Factors

- **Correctness** − Correctness determines whether the software requirements are appropriately met.
- **Usability** − Usability determines whether the software can be used by different categories of users (beginners, non-technical, and experts).
- **Portability** − Portability determines whether the software can operate in different platforms with different hardware devices.
- **Maintainability** − Maintainability determines the ease at which errors can be corrected and modules can be updated.
- **Reusability** − Reusability determines whether the modules and classes can be reused for developing other software products.

Object-Oriented Metrics

Metrics can be broadly classified into three categories: project metrics, product metrics, and process metrics.

Project Metrics

Project Metrics enable a software project manager to assess the status and performance of an ongoing project. The following metrics are appropriate for object-oriented software projects −

- Number of scenario scripts
- Number of key classes
- Number of support classes
- Number of subsystems

Product Metrics

Product metrics measure the characteristics of the software product that has been developed. The product metrics suitable for object-oriented systems are −

- **Methods per Class** − It determines the complexity of a class. If all the methods of a class are assumed to be equally complex, then a class with more methods is more complex and thus more susceptible to errors.
- **Inheritance Structure** − Systems with several small inheritance lattices are more well–structured than systems with a single large inheritance lattice. As a thumb rule, an inheritance tree should not have more than 7 (± 2) number of levels and the tree should be balanced.
- **Coupling and Cohesion** − Modules having low coupling and high cohesion are considered to be better designed, as they permit greater reusability and maintainability.
- **Response for a Class** − It measures the efficiency of the methods that are called by the instances of the class.

Process Metrics

Process metrics help in measuring how a process is performing. They are collected over all projects over long periods of time. They are used as indicators for long-term software process improvements. Some process metrics are −

- Number of KLOC (Kilo Lines of Code)
- Defect removal efficiency
- Average number of failures detected during testing
- Number of latent defects per KLOC

IMPACT OF OBJECT ORIENTATION ON TESTING **P-7**

Traditional testing methods are not directly applicable to OO programs as they involve OO concepts including encapsulation, inheritance, and polymorphism. These concepts lead to issues, which are yet to be resolved. Some of these issues are listed below.

1.)Basic unit of unit testing.

- The class is natural unit for unit test case design.
- The methods are meaningless apart from their class.
- Testing a class instance (an object) can validate a class in isolation.
- When individually validated classes are used to create more complex classes in an application system, the entire subsystem must be tested as whole before it can be considered to be validated(integration testing).

2.)Implication of Encapsulation.

- Encapsulation of attributes and methods in class may create obstacles while testing. As methods are invoked through the object of corresponding class, testing cannot be accomplished without object.

- In addition, the state of object at the time of invocation of method affects its behavior. Hence, testing depends not only on the object but on the state of object also, which is very difficult to acquire.

3.) Implication of Inheritance.

- Inheritance introduce problems that are not found in traditional software.
- Test cases designed for base class are not applicable to derived class always (especially, when derived class is used in different context). Thus, most testing methods require some kind of adaptation in order to function properly in an OO environment.

4.)Implication of Genericity.

- Genericity is basically change in underlying structure.
- We need to apply white box testing techniques that exercise this change.

  i.)Parameterization may or may not affect the functionality of access methods.

  ii.)In Tableclass, elemType may have little impact on implementations of the access methods of the Table. Example: generic (parameterized class) class Tableclass(elemType) int numberelements; create(); insert(elemType entry); delete(elemType entry); isEmpty() returns boolean; isentered(elemType entry) returns boolean; endclass;

- But UniqueTable class would need to evaluate the equivalence of elements and this could vary depending on the representation of elemType. Example: class UniqueTable extends Table insert(elemType entry); endclass;

4.)Implications of Polymorphism

- Each possible binding of polymorphic component requires a seperate set of test cases.
- Many server classes may need to be integrated before a client class can be tested.
- It is difficult to determine such bindings.
- It complicates the integration planning and testing.

5.)Implications for testing processes

- Here we need to re-examine all testing techniques and processes.

In object-oriented (OO) paradigm, software engineers identify and specify the objects and services provided by each object. In addition, interaction of any two objects and constraints on each identified object are also determined. The main advantages of OO paradigm include increased reusability, reliability, interoperability, and extendibility.

With the adoption of OO paradigm, almost all the phases of software development have changed in their approach, environments, and tools. Though OO paradigm helps make the designing and development of software easier, it may pose new kind of problems. Thus, testing of software developed using OO paradigm has to deal with the new problems also. Note that object-oriented testing can be used to test the object-oriented software as well as conventional software.

OO program should be tested at different levels to uncover all the errors. At the algorithmic level, each module (or method) of every class in the program should be tested in isolation. For this, white-box testing can be applied easily. As classes form the main unit of object-oriented program, testing of classes is the main concern while testing an OO program. At the class level, every class should be tested as an individual entity. At this level, programmers who are involved in the development of class conduct the testing. Test cases can be drawn from requirements specifications, models, and the language used. In addition, structural testing methods such as boundary value analysis are extremely used. After performing the testing at class level, cluster level testing should be performed. As classes are collaborated (or integrated) to form a small subsystem (also known as cluster), testing each cluster individually is necessary. At this level, focus is on testing the components that execute concurrently as well as on the interclass interaction. Hence, testing at this level may be viewed as integration testing where units to be integrated are classes. Once all the clusters in the system are tested, system level testing begins. At this level, interaction among clusters is tested.

Usually, there is a misconception that if individual classes are well designed and have proved to work in isolation, then there is no need to test the interactions between two or more classes when they are integrated. However, this is not true because sometimes there can be errors, which can be detected only through integration of classes. Also, it is possible that if a class does not contain a bug, it may still be used in a wrong way by another class, leading to system failure.

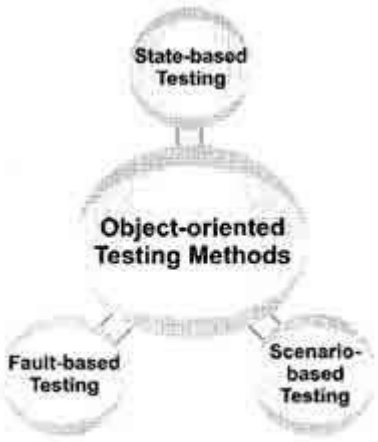
Developing Test Cases in Object-oriented Testing

The methods used to design test cases in OO testing are based on the conventional methods. However, these test cases should encompass special features so that they can be used in the object-oriented environment. The points that should be noted while developing test cases in an object-oriented environment are listed below.

1. It should be explicitly specified with each test case which class it should test.
2. Purpose of each test case should be mentioned.

3. External conditions that should exist while conducting a test should be clearly stated with each test case.
4. All the states of object that is to be tested should be specified.
5. Instructions to understand and conduct the test cases should be provided with each test case.

Object-oriented Testing Methods



Object-oriented Testing Methods

State-based testing is used to verify whether the methods (a procedure that is executed by an object) of a class are interacting properly with each other. This testing seeks to exercise the transitions among the states of objects based upon the identified inputs.

For this testing, finite-state machine (FSM) or state-transition diagram representing the possible states of the object and how state transition occurs is built. In addition, state-based testing generates test cases, which check whether the method is able to change the state of object as expected. If any method of the class does not change the object state as expected, the method is said to contain errors.

To perform state-based testing, a number of steps are followed, which are listed below.

1. Derive a new class from an existing class with some additional features, which are used to examine and set the state of the object.
2. Next, the test driver is written. This test driver contains a main program to create an object, send messages to set the state of the object, send messages to invoke methods of the class that is being tested and send messages to check the final state of the object.
3. Finally, stubs are written. These stubs call the untested methods.

Fault-based Testing **P-9**

Fault-based testing is used to determine or uncover a set of plausible faults. In other words, the focus of tester in this testing is to detect the presence of possible faults. Fault-based testing starts by examining the analysis and design models of OO software as these models may provide an idea of problems in the implementation of software. With the knowledge of system under test and experience in the application domain, tester designs test cases where each test case targets to uncover some particular faults.

The effectiveness of this testing depends highly on tester experience in application domain and the system under test. This is because if he fails to perceive real faults in the system to be plausible, testing may leave many faults undetected. However, examining analysis and design models may enable tester to detect large number of errors with less effort. As testing only proves the existence and not the absence of errors, this testing approach is considered to be an effective method and hence is often used when security or safety of a system is to be tested.

Integration testing applied for OO software targets to uncover the possible faults in both operation calls and various types of messages (like a message sent to invoke an object). These faults may be unexpected outputs, incorrect messages or operations, and incorrect invocation. The faults can be recognized by determining the behavior of all operations performed to invoke the methods of a class.

Scenario-based Testing

Scenario-based testing is used to detect errors that are caused due to incorrect specifications and improper interactions among various segments of the software. Incorrect interactions often lead to incorrect outputs that can cause malfunctioning of some segments of the software. The use of scenarios in testing is a common way of describing how a user might accomplish a task or achieve a goal within a specific context or environment. Note that these scenarios are more context- and user specific instead of being product-specific. Generally, the structure of a scenario includes the following points.

1. A condition under which the scenario runs.
2. A goal to achieve, which can also be a name of the scenario.
3. A set of steps of actions.
4. An end condition at which the goal is achieved.
5. A possible set of extensions written as scenario fragments.

Scenario- based testing combines all the classes that support a use-case (scenarios are subset of use-cases) and executes a test case to test them. Execution of all the test cases ensures that all methods in all the classes are executed at least once during testing. However, testing all the objects (present in the classes combined together) collectively is difficult. Thus, rather than

testing all objects collectively, they are tested using either top-down or bottom-up integration approach.

This testing is considered to be the most effective method as scenarios can be organized in such a manner that the most likely scenarios are tested first with unusual or exceptional scenarios considered later in the testing process. This satisfies a fundamental principle of testing that most testing effort should be devoted to those paths of the system that are mostly used.

Challenges in Testing Object-oriented Programs

Traditional testing methods are not directly applicable to OO programs as they involve OO concepts including encapsulation, inheritance, and polymorphism. These concepts lead to issues, which are yet to be resolved. Some of these issues are listed below.

Encapsulation of attributes and methods in class may create obstacles while testing. As methods are invoked through the object of corresponding class, testing cannot be accomplished without object. In addition, the state of object at the time of invocation of method affects its behavior. Hence, testing depends not only on the object but on the state of object also, which is very difficult to acquire.

Inheritance and polymorphism also introduce problems that are not found in traditional software. Test cases designed for base class are not applicable to derived class always (especially, when derived class is used in different context). Thus, most testing methods require some kind of adaptation in order to function properly in an OO environment.

PART – A

1. What is Object Oriented Methodology?

Object Oriented Methodology asks the analyst to determine

- ✓ What the objects of the system are?
- ✓ What responsibilities and relationships an object has to do with the other objects?
- ✓ How they behave over time?

2. Mention three types of Object Oriented Methodologies.

1. Object Modeling Techniques (OMT)
2. Object Process Methodology (OPM)
3. Rational Unified Process (RUP)

3. List out the processes involved in OPM.

It consists of three main processes:

I. Initiating: It determines high level requirements, the scope of the system and the resources that will be required.

II. Developing: It involves the detailed analysis, design and implementation of the system.

III. Deploying: It introduces the system to the user and subsequent maintenance of the system.

4. What are the phases involved in RUP?

It consists of four phases which can be broken down into iterations.

- ✓ Inception
- ✓ Elaboration
- ✓ Construction
- ✓ Transition

5. What is the objective of OO Methodology?

Objectives of Object Oriented Methodologies

- To encourage greater re-use.
- To produce a more detailed specification of system constraints.
- To have fewer problems with validation (Are we building the right product?).

6. List the benefits of Object Oriented Methodologies.

1. It represents the problem domain, because it is easier to produce and understand designs.
2. It allows changes more easily.
3. It provides nice structures for thinking, abstracting and leads to modular design.
4. Simplicity
5. Reusability
6. Increased Quality
7. Maintainable
8. Scalable
9. Modularity
10. Modifiability
11. Client/Server Architecture

7. What is Software Quality Assurance?

Software quality assurance is a methodology that determines the extent to which a software product is fit for use.

8. What Quality Factors are used to measure software?

1. Correctness
2. Usability
3. Portability
4. Maintainability
5. Reusability

9. Mention how OO metrics are classified.

Object-Oriented Metrics

Metrics can be broadly classified into three categories: project metrics, product metrics, and process metrics.

10. List some process metrics used in software measurement.

Process metrics are −

- Number of KLOC (Kilo Lines of Code)
- Defect removal efficiency
- Average number of failures detected during testing
- Number of latent defects per KLOC

11. What are the issues faced during OO testing?[Nov 2015]

1. Basic unit of unit testing.
2. Implication of Encapsulation.
3. Implication of Inheritance.
4. Implication of Genericity.
5. Implications of Polymorphism
6. Implications for testing processes

Define Test case. Give example [Nov 2017]

PART – B

1. **What is OO Methodology? Explain various OO methodologies used in software development.**
2. **Explain in detail about various issues in OO testing. [May 2018][May 2017][May 2019]**
3. **What is OO testing? Explain in detail about concepts of OO testing in OOAD.[Nov 2015]**
4. **Expalin in detail about software quality assurance.**
5. **What is test case? How test case is developed and testing is planned?**