❖ Testing Object Oriented Software Introduction to OO testing concepts:-

Prerequisite – Software Testing
Software typically undergoes many levels of testing, from unit testing to system or acceptance testing. Typically, in-unit testing, small "units", or modules of the software, are tested separately with focus on testing the code of that module. In higher, order testing (e.g, acceptance testing), the entire system (or a subsystem) is tested with the focus on testing the functionality or external behavior of the system.

As information systems are becoming more complex, the object-oriented paradigm is gaining popularity because of its benefits in analysis, design, and coding. Conventional testing methods cannot be applied for testing classes because of problems involved in testing classes, abstract classes, inheritance, dynamic binding, message, passing, polymorphism, concurrency, etc.
Testing classes is a fundamentally different problem than testing functions. A function (or a procedure) has a clearly defined input-output behavior, while a class does not have an input-output behavior specification. We can test a method of a class using approaches for testing functions, but we cannot test the class using these approaches.

According to Davis the dependencies occurring in conventional systems are:

- Data dependencies between variables
- Calling dependencies between modules
- Functional dependencies between a module and the variable it computes
- Definitional dependencies between a variable and its types.

But in Object-Oriented systems there are following additional dependencies:

- Class to class dependencies
- Class to method dependencies
- Class to message dependencies
- Class to variable dependencies
- Method to variable dependencies
- Method to message dependencies
- Method to method dependencies

Issues in Testing Classes:
Additional testing techniques are, therefore, required to test these dependencies. Another issue of interest is that it is not possible to test the class dynamically, only its instances i.e, objects can be tested. Similarly, the concept of inheritance opens various issues e.g., if changes are made to a parent class or superclass, in a larger system of a class it will be difficult to test subclasses individually and isolate the error to one class.

In object-oriented programs, control flow is characterized by message passing among objects, and the control flow switches from one object to another by inter-object communication. Consequently, there is no control flow within a class like functions. This lack of sequential control flow within a class requires different approaches for testing. Furthermore, in a function, arguments passed to the function with global data determine the path of execution within the procedure. But, in an object, the state associated with the object also influences the path of execution, and methods of a class can communicate among themselves through this state because this state is persistent across invocations of methods. Hence, for testing objects, the state of an object has to play an important role.

Techniques of object-oriented testing are as follows:

1. **Fault Based Testing:**
   This type of checking permits for coming up with test cases supported the consumer specification or the code or both. It tries to identify possible faults (areas of design or code that may lead to errors.). For all of these faults, a test case is developed to "flush" the errors out. These tests also force each time of code to be executed.
   This method of testing does not find all types of errors. However, incorrect specification and interface errors can be missed. These types of errors can be uncovered by function testing in the traditional testing model. In the object-oriented model, interaction errors can be uncovered by scenario-based testing. This form of Object oriented-testing can only test against the client's specifications, so interface errors are still missed.
2. **Class Testing Based on Method Testing:**
   This approach is the simplest approach to test classes. Each method of the class performs a well defined cohesive function and can, therefore, be related to unit testing of the traditional testing techniques. Therefore all the methods of a class can be involved at least once to test the class.
3. **Random Testing:**
   It is supported by developing a random test sequence that tries the minimum variety of operations typical to the behavior of the categories
4. **Partition Testing:**
   This methodology categorizes the inputs and outputs of a category so as to check them severely. This minimizes the number of cases that have to be designed.
5. **Scenario-based Testing:**
   It primarily involves capturing the user actions then stimulating them to similar actions throughout the test.
   These tests tend to search out interaction form of error.

❖ Difference Between Object-Oriented Testing and Conventional Testing

	Object-Oriented Testing	Conventional Testing
1.	In object-oriented Testing, a class is considered as a unit.	In conventional testing, the module or subroutine, or procedure are considered as a unit.
2.	Here, we cannot test a single operation in isolation but rather as part of a class.	Here, a single operation of a procedure can be tested.
3.	It focuses on composition.	It focuses on decomposition.
4.	It uses an incremental approach in the testing process.	It uses a sequential approach in the testing process.
5.	This testing requires at every class level wherein each class is tested individually.	This testing is following the waterfall life cycle in its testing process.
6.	This testing has a hierarchical control structure.	This testing does not have any hierarchical control structure.
7.	Top-down or bottom-up integration is possible in this testing.	Here, any ordering is not possible to follow.
8.	In object-oriented testing, it has unit, integration, validation, and system testing as its levels of testing.	Conventional Testing also has the same levels of testing but the approach is different.

❖ Issues in Object Oriented Testing:

Strong text : 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

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.

❖ Class Testing

The fundamental unit of an object-oriented program is a class. Class testing comprises those activities associated with verifying that the implementation of a class corresponds exactly with the specification for that class. If an implementation is correct, then each of the class's instances should behave properly.

Class testing is roughly analogous to unit testing in traditional testing processes and has many of the same problems that must be addressed (see sidebar). Class testing must also address some aspects of integration testing since each object defines a level of scope in which many methods interact around a set of instance attributes. Some of the most critical issues will be discussed in the context of concurrent issues in Chapter 8. The focus of this chapter is execution-based testing of classes. Our primary objective is to describe basic elements and strategies of testing classes, and we will focus on relatively simple classes. The testing of more complex classes will be addressed in the next two chapters.

We assume that a class to be tested has a complete and correct specification, and that it has been tested within the context of the models.[1] We assume the specification is expressed in a specification language such as the Object Constraint Language (OCL) [WK99] or a natural language, and/or as a state transition diagram. If more than one form of specification is used for a class, we assume all forms are consistent and that information may be taken from whichever form is most useful as the basis for developing test cases for the class. We prefer to use the most formal specification for generating test cases.

[1] Consistency is primarily a design consideration. When class testing is underway, design for the class should be finished at least as far as the current development iteration is concerned.

Ways to Test a Class
The code for a class can be tested effectively by review or by executing test cases. Review is a viable alternative to execution-based testing in some cases, but has two disadvantages over execution-based testing:

- Reviews are subject to human error.
- Reviews require considerably more effort with respect to regression testing, often requiring almost as many resources as the original testing.

❖What is GUI Testing?

GUI Testing is a software testing type that checks the Graphical User Interface of the Software. The purpose of Graphical User Interface (GUI) Testing is to ensure the functionalities of software application work as per specifications by checking screens and controls like menus, buttons, icons, etc.
GUI is what the user sees. Say if you visit guru99.com what you will see say homepage it is the GUI (graphical user interface) of the site. A user does not see the source code. The interface is visible to the user. Especially the focus is on the design structure, images that they are working properly or not.

Need of GUI Testing

Now the basic concept of GUI testing is clear. The few questions that will strike in your mind will be

- Why do GUI testing?
- Is it really needed?
- Does testing of functionally and logic of Application is not more than enough?? Then why to waste time on UI testing.

To get the answer to think as a user, not as a tester. A user doesn't have any knowledge about XYZ software/Application. It is the UI of the Application which decides that a user is going to use the Application further or not.

A normal User first observes the design and looks of the Application/Software and how easy it is for him to understand the UI. If a user is not comfortable with the Interface or find Application complex to understand he would never going to use that Application Again. That's why, GUI is a matter for concern, and proper testing should be carried out in order to make sure that GUI is free of Bugs.

What do you Check-in GUI Testing?

The following checklist will ensure detailed GUI Testing in Software Testing.

- Check all the GUI elements for size, position, width, length, and acceptance of characters or numbers. For instance, you must be able to provide inputs to the input fields.
- Check you can execute the intended functionality of the application using the GUI
- Check Error Messages are displayed correctly
- Check for Clear demarcation of different sections on screen
- Check Font used in an application is readable
- Check the alignment of the text is proper
- Check the Color of the font and warning messages is aesthetically pleasing
- Check that the images have good clarity
- Check that the images are properly aligned
- Check the positioning of GUI elements for different screen resolution.

GUI Testing Techniques

GUI Testing Techniques can be categorized into three parts:

Manual Based Testing

Under this approach, graphical screens are checked manually by testers in conformance with the requirements stated in the business requirements document.

Record and Replay

GUI testing can be done using automation tools. This is done in 2 parts. During Record, test steps are captured by the automation tool. During playback, the recorded test steps are executed on the Application Under Test. Example of such tools – QTP.

Model Based Testing

A model is a graphical description of a system's behavior. It helps us to understand and predict the system behavior. Models help in a generation of efficient test cases using the system requirements. The following needs to be considered for this model based testing:

- Build the model
- Determine Inputs for the model
- Calculate the expected output for the model
- Run the tests
- Compare the actual output with the expected output
- A decision on further action on the model

❖ Object Oriented Integration and System Testing

Testing Object-Oriented Systems

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, Integration testing, and system testing.

- **Unit Testing:** In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free. Unit testing is the responsibility of the application engineer who implements the structure.
- **Integration Testing:** This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside. Subsystem tests can be used as regression tests for each newly released version of the subsystem.
- **System Testing:** System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.

Object-Oriented Testing Techniques

Grey Box Testing

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are −

State model based testing − This encompasses state coverage, state transition coverage, and state transition path coverage.

- **Use case based testing** − Each scenario in each use case is tested.
- **Class diagram based testing** − Each class, derived class, associations, and aggregations are tested.
- **Sequence diagram based testing** − The methods in the messages in the sequence diagrams are tested.

Techniques for Integration Testing

The two main approaches of Integration testing are −

- **Thread based testing** − All classes that are needed to realize a single use case in a subsystem are integrated and tested.
- **Use based testing** − The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

Categories of System Testing

- **Alpha testing** − This is carried out by the testing team within the organization that develops software.
- **Beta testing** − This is carried out by select group of co-operating customers.
- **Acceptance testing** − This is carried out by the customer before accepting the deliverables.