Object Oriented Testing in Software Testing

SoftwareTesting

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 a 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 a 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 issue 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.

- 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 the 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.

Purpose of Object Oriented Testing

1. **Object Interaction Validation:** Check to make sure objects interact with one another appropriately in various situations. Testing makes ensuring that the interactions between objects in object-oriented systems result in the desired results.
2. **Determining Design Errors:** Find the object-oriented design's limitations and design faults. Testing ensures that the design complies with the desired architecture by assisting in the identification of problems with inheritance, polymorphism, encapsulation and other OOP concepts.

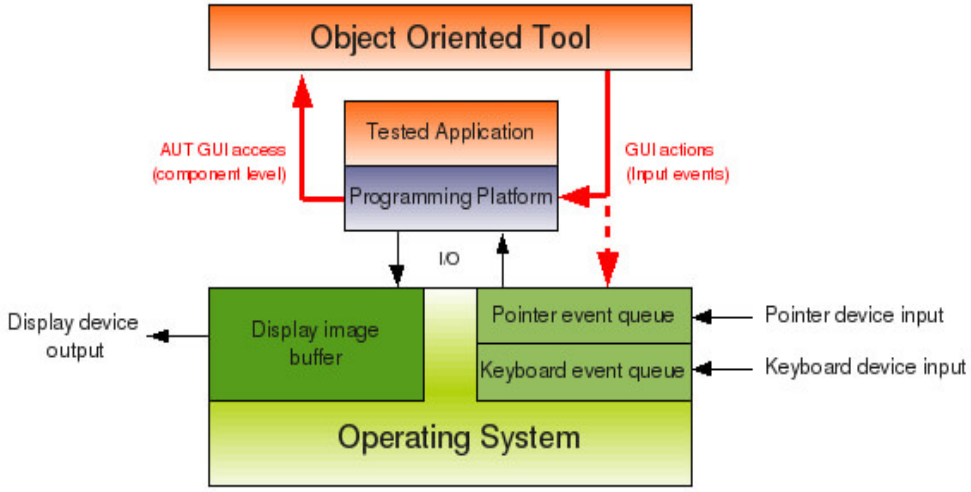
3. **Finding Integration Problems:** Evaluate an object's ability to integrate and communicate with other objects when it is part of a bigger component or subsystem. This helps in locating integration difficulties, such improper method calls or issues with data exchange.
4. **Assessment of Reusable Code:** Evaluate object-oriented code's reusability. Code reuse is promoted by object-oriented programming via features like inheritance and composition. Testing ensures that reusable parts perform as intended in various scenarios.
5. **Verification of Handling Exceptions**: Confirm that objects respond correctly to error circumstances and exceptions. The purpose of object-oriented testing is to make sure that the software responds carefully and is durable in the face of unforeseen occurrences or faults.
6. **Verification of Uniformity:** Maintain uniformity inside and between objects and the object-oriented system as a whole. Maintainability and readability are enhanced by consistency in naming standards, coding styles and compliance to design patterns.

Difference between Object-Oriented Testing and Conventional Testing

This article will provide you a detailed comparison between Object-oriented testing and Conventional Testing. Let's start with the brief introduction of object-oriented testing.

What is Object-Oriented Testing?

Object-oriented testing is a type of software testing that focuses on verifying the behaviour of individual objects or classes in an object-oriented system. The goal of object-oriented testing is to ensure that each object or class in the system performs its functions correctly and interacts properly with other objects or classes.



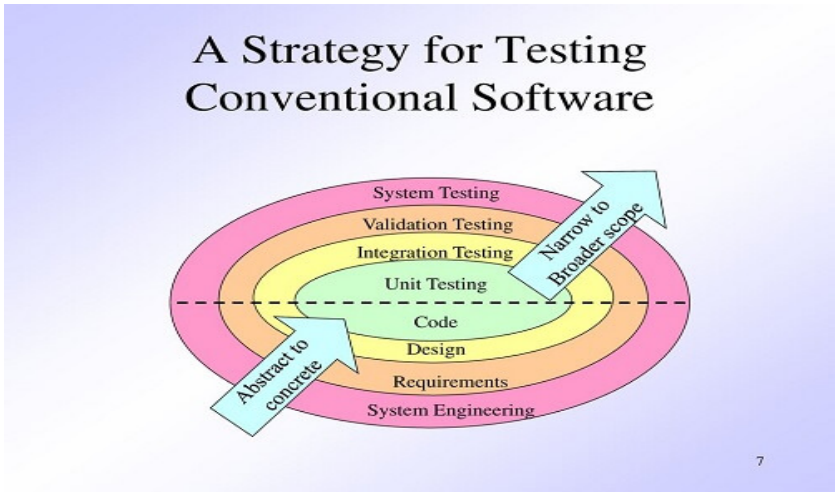
Object-oriented programming emphasises the use of objects and classes to organise and structure software, and object-oriented testing is built on these ideas. In object-oriented testing, the behaviour of an object or class is tested by creating test cases that simulate different scenarios or inputs that the object or class might encounter in the real world.

Unit testing, integration testing, and system testing are common testing phases in object-oriented testing. Unit testing focuses on testing individual objects or classes in isolation, while integration testing verifies that different objects or classes can work together as expected. System testing examines every component of the system, including all of its classes and objects.

Object-oriented testing can be challenging, as it requires a thorough understanding of the system's design and implementation, as well as the ability to create effective test cases that cover all possible scenarios. The system is made to be dependable, effective, and simple to manage, hence it is a crucial step in the software development process.

What is Conventional Testing?

Conventional testing, commonly referred to as traditional testing, is a type of software testing that focuses on comparing the functionality of a software system or application to a set of established standards or criteria. In conventional testing, the software is typically tested by executing a series of pre-written test cases that are designed to cover all of the specified requirements or features of the software.



Testing at several levels, such as unit testing, integration testing, system testing, and acceptance testing, is a common practise in conventional testing. Unit testing focuses on testing individual components or modules of the software in isolation, while integration testing verifies that different components or modules can work together as expected. In contrast to acceptance testing, which involves evaluating the programme with end users to make sure it satisfies their needs and requirements, system testing examines the entire software system as a whole, including its interfaces with other systems.

Conventional testing is often manual, meaning that human testers execute the test cases and evaluate the results. To speed up testing and lower the chance of human error, automated testing technologies are, however, being employed more and more in conventional testing.

As it offers an organised method of testing that helps to verify the software is dependable, functional, and fits the demands of its users, conventional testing is frequently employed in the creation of software. However, conventional testing has limitations, as it can be time-consuming, expensive, and may not detect all types of software defects or issues. As a result, newer testing approaches such as agile and DevOps are becoming increasingly popular, which focus on continuous testing, integration, and delivery of software.

Difference between Object-oriented testing and Conventional Testing

Here are 14 key differences between Object-oriented testing and Conventional Testing

1	This emphasises performing isolated testing on certain objects or classes	This emphasises testing a software system's functionality against predetermined criteria or requirements

2	It verifies the behaviour of each object or class in the system.	It verifies the behaviour of the entire software system.
3	Tests the interactions between objects or classes.	Tests the interactions between software components or modules.
4	It uses mock objects to simulate the behaviour of dependent objects or classes.	It does not use mock objects.
5	It can be more time-consuming than conventional testing.	It can be faster than object-oriented testing.
6	Requires a thorough understanding of the system's design and implementation.	Requires a thorough understanding of the system's requirements and specifications.
7	Involves testing at multiple levels, including unit testing, integration testing, and system testing.	Involves testing at multiple levels, including unit testing, integration testing, system testing, and acceptance testing.
8	Can be more complex than conventional testing.	Can be simpler than object-oriented testing.
9	Focuses on testing the individual behaviour of objects or classes.	Focuses on testing the overall behaviour of the software system.
10	Can detect defects or issues that may not be detected by conventional testing.	May not detect all types of software defects or issues.
11	Can be more effective in testing complex object-oriented systems.	May be less effective in testing complex object-oriented systems.
12	Involves creating test cases that simulate different scenarios or inputs that the object or class might encounter in the real world.	Involves creating test cases that cover all possible scenarios or requirements of the software.
13	Requires the ability to create effective test cases that cover all possible scenarios.	Requires the ability to write test cases that cover all predetermined requirements or

		specifications.
14	May require the use of specialized testing tools and frameworks.	May not require specialized testing tools and frameworks.