Testing Object-Oriented

- 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.

1. Data dependencies between variables
2. Calling dependencies between modules
3. Functional dependencies between a module and the variable it computes
4. 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:

1. **Fault Based Testing:** This type of checking permits for coming up with test cases supporting 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 forms 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 ensures 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.