# UNIT 5
# TESTING

## 5.1 OBJECT ORIENTED MEHODOLOGIES

### The Essentials

Many methodologies are available to choose from for the system development. Each methodology is based on modeling the business problem and implementation in an object oriented fashion. The Rumbaugh et al method has a strong method for producing object models. Jacobsonet al have a strong method for producing user-driven requirement and object oriented analysis model. Booch has a strong method for producing detailed object oriented design models

### Rumbaugh's Object Modeling Technique:

- 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:-

Object model describes the structure of objects in a system, their identity and relationships to other objects, attributes, and operations.

The object model is represented graphically with an object diagram.

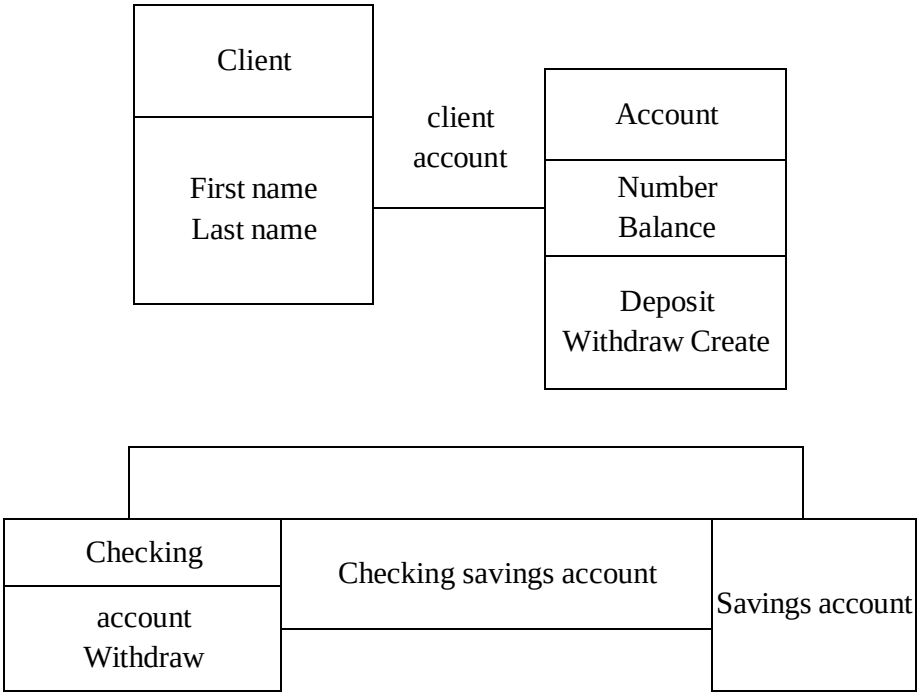


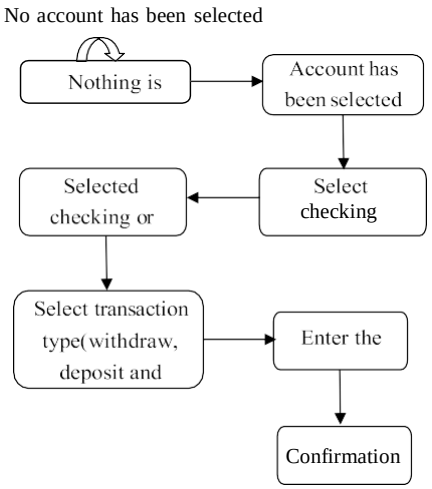

***Figure 5.1 OMT object model of a bank system***

Boxes represent classes

Filled represents specialization

## The OMT Dynamic Model:

- OMT provides a detailed and comprehensive dynamic model
- The OMT state transition diagram is a network of states and events.
  - Each state receives one or more events, at which it makes the transition to the next state.
  - The next state depends on the current state as well as the events.



State transition diagrams for the bank application user interface.

- Round boxes represents states
- Arrows represents transitions

***The OMT Functional Model***

- The OMT data flow diagram (DFD) shows the flow of data between different processing in a business.
- Data Flow Diagrams use four primary symbols:
  - The *process* is any function being performed
  - The *data flow* shows the direction of data element movement
  - The *data store* is a location where data are stored.
  - An *external entity* is a source or destination of a data element.

## The Booch Methodology :

- 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.

- ***Design or create the system architecture.***

  Use the class diagram to decide what classes exist and how they relate to each other, the object diagram to decide what mechanisms are used, the module diagram to map out where each class and object should be declared, and the process diagram to determine to which processor to allocate a process.

- ***Evolution or implementation.***

  Refine the system through much iteration.

- ***Maintenance.***

  Make localized changes to the system to add new requirements and eliminate bugs.

## Micro development process.

- The micro development process is a description of the day-to-day activities.
- Steps involved:
  - Identify classes and objects.
  - Identify classes and object semantics.
  - Identify classes and object relationships.
  - Identify classes and object interfaces and implementation.

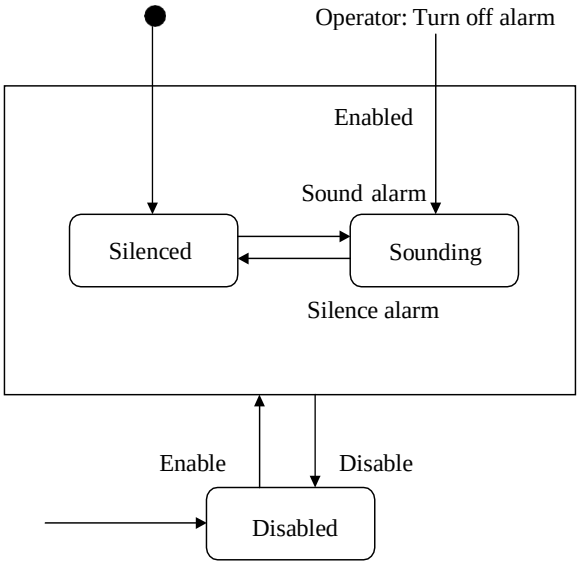



***Figure 5.2 Alarm fixed***

  - An alarm state transition diagram with Booch notation

## 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

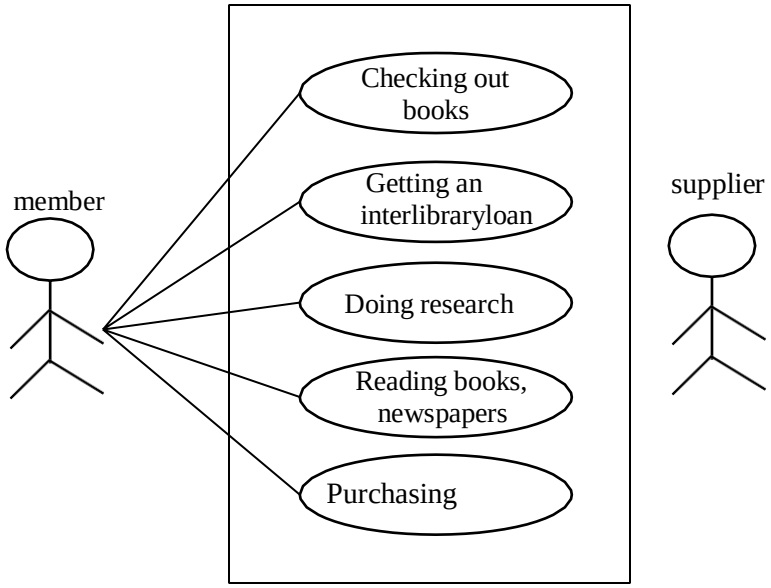
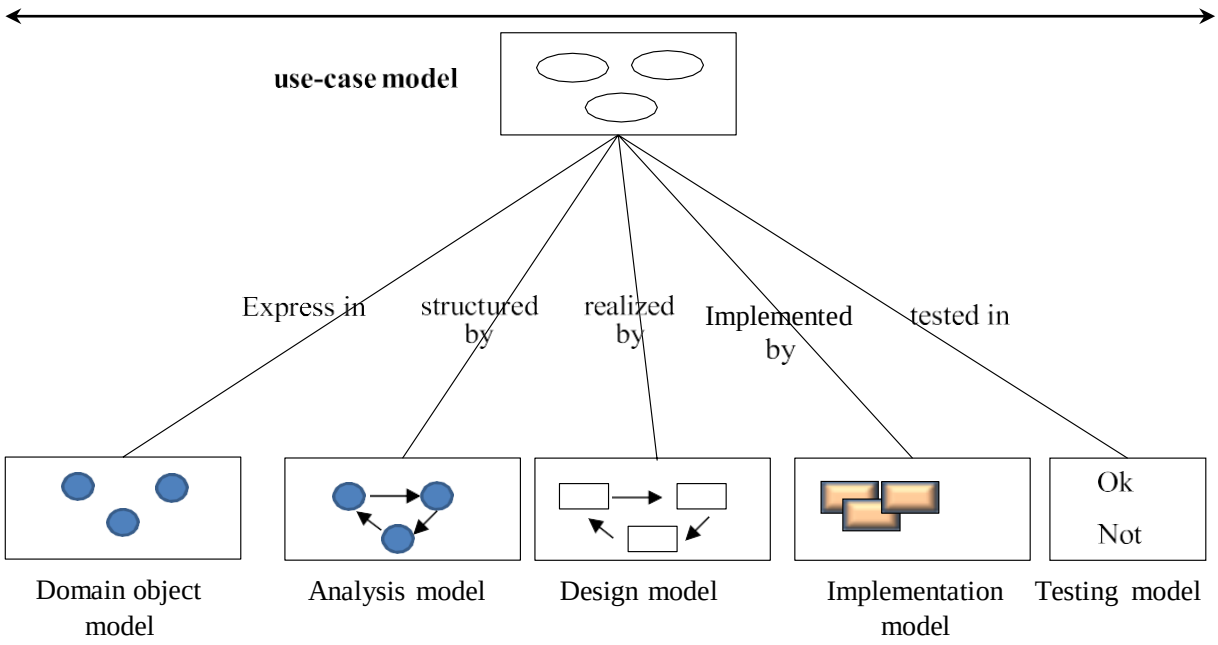



***Figure 5.3 Library***

- 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:-
  - The implementation environment must be identified for the design model.

- Testing phase:-
  - This level includes unit testing, integration testing and system testing.

**Patterns and the Various Pattern Templates:**

- ***Pattern –***
  - Identifies a common structure that makes it useful for design.
  - provide common vocabulary
  - provide“shorthand” for effectively communicating complex principles
  - help document software architecture
  - capture essential parts of a design in compact form
  - show more than one solution
  - describe software abstractions
- ***Patterns do not...***
  - provide an exact solution
  - solve all design problems
  - only apply for object-oriented design
- Involves a general description of a solution to a recurring problem.
- ***Properties of a good pattern:***
  - It solves a problem:-
    - Patterns capture solutions not just abstract principles or strategies
  - It is a proven concept:-
    - Patterns capture solutions with a track record, not theory or speculation
  - Solution is not obvious:-
    - The best patterns generate a solution to a problem indirectly
  - Describes a relationship:-
    - Patterns do not just describes modules but describes deeper system structures and mechanisms
  - Has significant human component:-
    - All software serves human comfort or quality of life
- ***Generative patterns*** – Tells us how to generate something.
  - observed in a system
  - descriptive and passive

- ***Non generative patterns –***
  - ogenerate systems or pats of systems
  - operspective and active
- ***Patterns templates.***
  - *Name:-*
    - A meaningful name allows us to use a single word or short phrase to refer to the pattern and the knowledge and structure it describes.
    - Some pattern forms also provide a classification of the pattern in addition to its name.
  - *Problem:-*
    - A statement of the problem that describes its intent: the goals and objectives it wants to reach within the given context and forces.
    - The forces oppose these objectives as well as each other.
  - *Context:-*
    - The preconditions under which the problem and its solution seem to recur and for which the solution is desirable.
    - It can be thought of as the initial configuration of the system before the pattern is applied to it.
  - *Forces:-*
    - A description of the relevant forces and constraints and how they interact or with one another and with the goals that the user wish to achieve.
    - A concrete scenario that serves as the motivation for the pattern frequently is employed.
  - *Solution:-*
    - Static relationships and dynamic rules describing how to realize the desired outcome.
    - It describes how to construct the necessary products.
    - It encompasses the pictures, diagrams and prose that identify the pattern structure, and their participants and collaborations to show how the problem is solved.
  - *Examples:-*
    - One or more sample applications of the pattern that illustrate a specific initial context; how the pattern is applied to and transforms that context.
  - *Resulting context:-*
    - The state or configuration of the system after the pattern has been applied, including the consequences of applying the pattern and other problems and patterns that may arise from the new context.

- *Rationale:-*
  - Steps or rules in the pattern and also of the pattern as a whole in terms of how and why it resolves it forces in a particular way to be in alignment with desired goals.

- ***Related patterns:-***
  - The static and dynamic relationships between this pattern and others within the same pattern language or system.
  - Related pattern often share common forces.
  - They also frequently have an initial or resulting context that is compatible with the resulting or initial context of another pattern.
- ***Known uses:-***
  - The known occurrences of the pattern and its application within existing systems.

This helps to validate a pattern by verifying that it indeed is a proven solution to the recurring problem.

## FRAMEWORKS:

Frameworks are a way of delivering application development patterns to support best practice sharing during application development .

A frame work is a way of presenting a generic solution to a problem that can be applied to all levels in a development. Several design patterns in fact a framework can be viewed as the implementation of a system of design patterns.

The major differences between design patterns and frameworks as follows

- Design patterns are more abstract than frameworks.
- Design patterns are smaller architectural elements than frameworks.
- Design patterns are less specialized than frameworks.

## The Unified Approach:

- Establishes a unifying and unitary framework by utilizing UML.
- The processes are:
  - Use case driven development.
  - Object oriented analysis.
  - Object oriented design.
  - Incremental development and prototyping
  - Continuous testing.
- Methods and technologies employed include:

- UML:- Unified Modeling approach is used for modeling
- Layered approach:-
- Repository:- Repository for object-oriented system development patterns and frameworks
- CBD:- Component based development
  - The Unified Approach allows iterative development by allowing to go back and forth between the design and the modeling or analysis phase.
  - It makes backtracking very easy and departs from the linear waterfall process, which allows no form of backtracking.

- Object oriented Analysis.
  - Identify actors.
  - Develop a simple process model.
  - Develop the use case.
  - Develop interaction diagrams.
  - Identify classes.
- Object oriented design
  - Design classes, attributes, methods etc.
  - Design the access layer.
  - Design the prototype user interface.
  - User satisfaction and usability tests.
  - Iterate and refine the design.
- Continuous testing
- UML – modeling language
- Repository
  - Allows maximum reuse of previous experience.
  - Should be accessible by many people.
- Layered approach
  - The business layer.
    - Displaying results.
    - Data access details
  - The user interface or view layer.
  - Responding to user interaction:-

        - It must be designed to translate actions by the user, such as clicking on a button or selecting from a menu.
    - Displaying business objects:-
        - This layer must paint a best possible picture of the business objects for the user.

## 5.2 QUALITY ASSURANCE TESTS: -

- Debugging is a process of finding out where something went wrong and correcting the code to eliminate the errors or bugs that cause unexpected results. A software debugging system can provide tools for finding errors in programs and correcting them.
- Kinds of errors: In general, a software has three types of errors such as below
    1) Language (syntax) errors are result of incorrectly constructed code, such as an incorrectly typed keyword or punctuations. They are easiest error to be detected on simple running system
    2) Run–time errors are detected on running, when a statement attempts an operation that is impossible to carry out. Eg.: if program tries to access a non-exist file or object, it occurs
    3) Logic errors occur when expected output is not formed. They can detected only by testing the code and analyzing the results performed by intended codes
- The elimination of syntactical bug is the process of debugging, whereas detection and elimination of logical bug is the process of testing. Quality assurance testing can be divided into two major categories: error–based testing and scenario–based testing
- Error–based testing techniques search a given class's method for particular clues of interests, then describe how these clues should be tested. E.g: Boundary condition testing
- Scenario–based testing also called usage–based testing, concentrates on capturing use –cases. Then it traces user's task, performing them with their variants as tests. It can identify interaction bugs. These are more complex tests tend to exercise multiple subsystems in a single test covering higher visibility system interaction bugs
- S/w testing is one element of a broader topic that is often referred to as verification and validation (V&V). Verification refers to set of activities that ensure that software correctly implements a specific function. Validation refers to different set of activities that ensure that s/w that hs been built is traceable to customer requirements

### Testing Strategies: -

- The objective of s/w testing is to uncover errors. The various testing strategies constitutes –
    - Unit Testing – Black Box testing, White black testing
    - Integration Testing – Top–down testing, Bottom–up testing, Regression testing

- Validation Testing – Alpha test, Beta test and
- System Testing – Recovery testing, Security testing, Stress testing, Performance testing o Tom Gilb argues following issues for successful s/w testing strategy is to be implemented:
  1) Specify product requirements in a quantifiable manner long before testing commences
  2) State testing objectives explicitly
  3) Understand the users of s/w and develop a profile for each user category
  4) Develop a testing plan that emphasizes "rapid cycle testing"
  5) Build "robust" s/w that is designed to test itself
  6) Use effective formal technical reviews as a filter prior to testing
  7) Conduct formal technical reviews to assess the test strategy and test cases themselves
  8) Develop a continuous improvement approach for testing process

- Unit and integration tests concentrate on functional verification of a module and incorporation of
- modules into a program structure. Validation testing demonstrates traceability to s/w requirements and system testing validates s/w once it has been incorporated into a larger system
  1) Unit test focuses verification effort on smallest unit of s/w design the module. It constitutes two inner types of testing – White box testing and Black box testing
     - Black box testing: The concept of black box is used to represent a system that's inside working are
     - not available for inspection. In black box testing, we try various inputs and examine resulting output though which we learn what the box does nor how conversion takes place
     - White box testing: White box testing assumes that specific logic important and must be tested to guarantee system's proper functioning. One form of white box testing called path testing, makes certain that each path in a object's method is executed at least once during testing & is of types
       1) Statement testing coverage: Its aim is to test every statement in object's method at least once
       2) Branch testing coverage: Idea is to perform enough tests ensuring all branches are perfect

- ***Integration Testing:*** It is a systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing. The object is to take unit tested modules and build a program structure that has been dictated by design. It has again 3 testing patterns: Top–down testing, Bottom – up testing and regression testing

- ***Top–down testing*:** It assumes that main logic or object interactions and system messages of application need more testing than an individual object's method or supporting logic. This strategy can detect serious flaws early in implementation
- ***Bottom–Up testing*:** It starts with details of system and proceeds to higher levels by a progressive aggregation of details until they collectively fit requirements for system i.e., start with methods and classes then to level up. It makes sense as it checks behavior of piece of code before it is used
- ***Regression testing*:** It is activity that helps ensure that changes (due to testing or other reasons) do not introduce unintended behavior or additional errors. The regression test suite contain three different classes of test cases:
  - A representative sample of tests that will exercise all s/w functions
  - Additional tests that focus on s/w functions that are likely to be affected by change
  - Tests that focus on s/w components that have been changed
- ***Validation Testing*:** After integration testing, s/w is completely assembled as a package: interfacing errors have been uncovered and corrected and final series of s/w tests – validating testing – begins. It is nothing but continuous execution on s/w by number of times & users. It is again implemented either by one of two methods: Alphatesting or Beta testing
- ***Alpha test*** is conducted in controlled environment at developer's site by a customer. The s/w is used in setting with developer "looking over shoulder" of user, recording errors & usage problems
- ***Beta test*** is conducted at one or more customer sites by end user(s) of the s/w. Its live application of s/w where customer records all problems that are encountered during beta testing and reports
- ***System Testing*:** It is a series of different tests whose primary purpose is to fully exercise the computer – based system. Although each test has a different purpose, all work to verify that all system elements have been properly integrated and perform allocated functions. The different tests in series are Recovery, security, stress and performance testing
- ***Recovery testing*** is a system test that forces s/w to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic, re – initialization, checkpoint mechanisms, data recovery and restart are each evaluated for correctness
- ***Security testing*** attempts to verify that protection mechanisms built into a system will in fact protect it from improper penetration. During this test, tester plays the role(s) of individual who desires to penetrate system. The role of system designer is to make penetration cost greater than value of information that will be obtained
- ***Stress testing*** executes a system in a manner that demands resources in abnormal quantity, frequency or volume. A variation of stress testing is a technique called sensitivity testing. In some situations, a very small range of data contained within the bunds of valid data for a programs may cause extreme and even erroneous processing or profound performance degradation

- ***Performance testing*** is designed to test run –time performance of s/w within context of an integrated system. It occurs throughout all steps in testing process. Performance tests are often coupled with stress testing and often require both h/w and s/w instrumentation

## 5.3 IMPACT OF OBJECT ORIENTATION ON TESTING

The impact of an object orientation on testing can be summarized as follows:

1) Some types of errors could become less plausible (not worth for testing)
2) Some types of errors could become more plausible (worth testing for now)
3) Some new types of errors might appear

Impact of Inheritance in Testing: If designers do not follow OOD guidelines especially, if test is done incrementally, it will lead with objects that are extremely hard to debug and maintain Reusability of Tests: Marick argues that simpler is a test, more likely it is to be reusable in sub classes. The models developed for analysis & design should be used for testing as well

## 5.4 DEVELOP TEST CASES AND TEST PLANS

### 5.4.1 Develop Test

- ***Myers*** describes the object of testing as follows:
  - Testing is process of executing a program with the intent of finding errors
  - A good test case is one that has a high probability of detecting an as–yet undiscovered error
  - A successful test case is one that detects as as–yet undiscovered error
- **Guidelines for developing Quality Assurance Test Cases**: Freedman & Thomas have developed guidelines that have been adapted for the Unified Approach
  - Describe which feature or service (external of internal), test attempts to cover
  - If test case is based on use case it must refer to use–case name and write test plan for that piece
  - Specify what to test on which method along with test feature and expected action
  - Test normal use of the object's methods
  - Test abnormal but reasonable use of the object's methods
  - Test abnormal and unreasonable use of object's methods
  - Test boundary conditions of number of parameters or input set of objects
  - Test object's interactions & message sent among them with assist of sequence diagram
  - On doing revision, document the cases so they become the starting basis for follow–up test

- Attempt to reach agreement on answers of what–if questions and repeat process until stabilized
- The internal quality of s/w such as its reusability and extendibility should be assessed as well

***Example****:* Testing a File Open feature, we specify the result as follows:

1) Drop down the File menu and select Open
2) Try opening following type of files* A file that is there (should work) * A file that is not

   there(should get an error message) * A file name with international characters (should work)

   * A file type that the program does not open (should get a message or conversion dialog box)

### 5.4.2 Test Plan : -

- A test is developed to detect and identify potential problems before delivering the s/w to its users. The test plan need not be very large; in fact, devoting too much time to the plan can be counter productive. The following steps are needed to create a plan
  1) **Objectives of test**: Create objectives of test and describe how to achieve them
  2) Development of test case: Develop test data, I/O based on domain of data & expected behavior
  3) Test analysis: It involves examination of test O/p and documentation of test results
- All passed tests should be repeated with revised program called regression testing. Most s/w companies use beta testing, a popular inexpensive, effective way to test s/w and alpha testing
- Guidelines for developing Test Plans: Thomas stated following guidelines for writing test plans
  - Specific appearance or format of test plan must include more details about test
  - It should contain a schedule and a list of required resources including number of peoples &time
  - Document every type of test planned with level of detail driven by several factors
  - A configuration control system provides a way of tracking changes to code should exist
  - Try to develop a habit of routinely bring test plan sync with product or product specification
  - At end of each moth or as reach each milestone, take time to complete routine updates

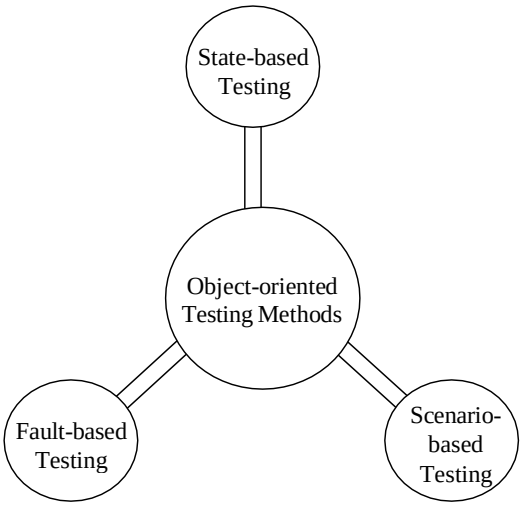
    1) The methods used to design test cases in OO testing are based on the conventional methods. However, these test cases should encompass special

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.

2) It should be explicitly specified with each test case which class it should test.
3) Purpose of each test case should be mentioned.
4) External conditions that should exist while conducting a test should be clearly stated with each test case.
5) All the states of object that is to be tested should be specified.
6) Instructions to understand and conduct the test cases should be provided with each test case.

## Object-oriented Testing Methods

As many organizations are currently using or targeting to switch to the OO paradigm, the importance of OO software testing is increasing. The methods used for performing object-oriented testing are discussed in this section.




***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***

1) 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 designmodels 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.
2) 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.

***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.

A condition under which the scenario runs.

A goal to achieve, which can also be a name of the scenario.

A set of steps of actions.

An end condition at which the goal is achieved.

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.