Object Oriented Testing

- ✓ Overview of Testing and object oriented Testing
- ✓ Types of Testing
- ✓ Unit testing
- ✓ Integrating testing
- ✓ System testing
- ✓ Object Oriented Testing strategies
- ✓ Test case design for OO software
- ✓ Inter class test Case design

Software Testing:

- Software testing is a process used to identify the correctness, completeness and quality of developed computer software.
- It is the process of executing a program / application under positive and negative conditions by manual or automated means.
- It checks for Specification, Functionality and Performance
- It is a validation and verification process.
- **Testing = Verification + Validation**
  Verification: Static testing (no run)
  Validation: Dynamic testing (code running)

Objectives of Software Testing:

- Uncover as many as errors (or bugs) as possible in a given product.
- Demonstrate a given software product matching its requirement specifications.
- Validate the quality of a software testing using the minimum cost and efforts.
- Generate high quality test cases, perform effective tests, and issue correct and helpful problem reports.

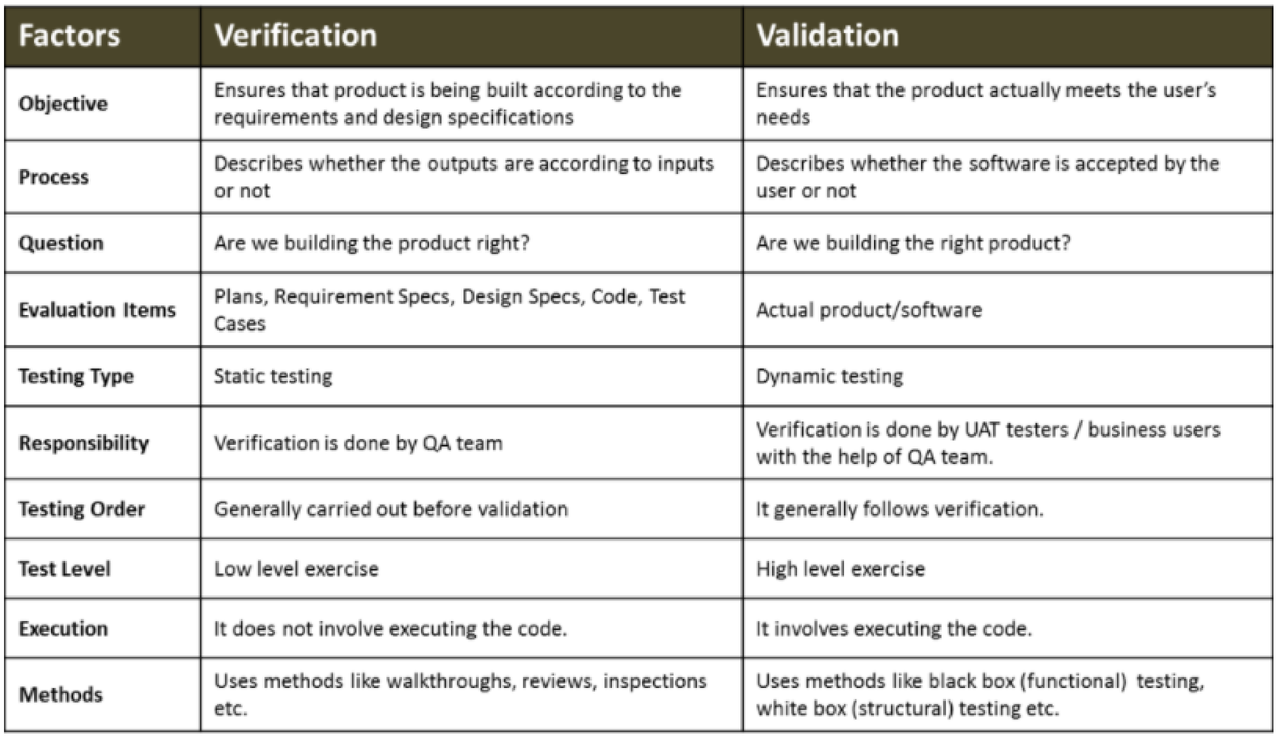
Verification:

- It is the process to make sure the product satisfies the conditions imposed at the start of the development phase.
- To make sure the product behaves the way we want it to.

Validation:

- Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase.
- To make sure the product is built as per customer requirements.

Difference between Verification and Validation



Factors	Verification	Validation
Objective	Ensures that product is being built according to the requirements and design specifications	Ensures that the product actually meets the user's needs
Process	Describes whether the outputs are according to inputs or not	Describes whether the software is accepted by the user or not
Question	Are we building the product right?	Are we building the right product?
Evaluation Items	Plans, Requirement Specs, Design Specs, Code, Test Cases	Actual product/software
Testing Type	Static testing	Dynamic testing
Responsibility	Verification is done by QA team	Verification is done by UAT testers / business users with the help of QA team.
Testing Order	Generally carried out before validation	It generally follows verification.
Test Level	Low level exercise	High level exercise
Execution	It does not involve executing the code.	It involves executing the code.
Methods	Uses methods like walkthroughs, reviews, inspections etc.	Uses methods like black box (functional) testing, white box (structural) testing etc.

Example of verification and validation

Now, let's take an example to explain verification and validation planning:

- In Software Engineering, consider the following specification for verification testing and validation testing,

A clickable button with name: Submet

- Verification would check the design doc and correcting the spelling mistake.
- Otherwise, the development team will create a button like

Fig: Example of Verification

- So new specification is

  A clickable button with name Submit

- Once the code is ready, Validation is done. A Validation test found:

Fig: Example of Validation

- Owing to Validation testing, the development team will make the submit button clickable.

Bug, Fault and Failure

“A person makes an **Error**
That creates a **fault** in the software
That can cause a **failure** in the operation”

- **Error:** An error is a human action that produces the incorrect result that results in a fault.
- **Bug:** The presence of error at the time of execution of the software.
- **Fault:** State of software caused by an error.
- **Failure:** Deviation of the software from its expected result. It is an event.
- **Test data:** A test data is the inputs which have been devised to test the system.
- **Test Case:** A test case is the triplet (I,0], where I is the data input to the system, S is the state of the system at which data is input, and 0 is the expected output of the system or Inputs to test the system and the predicted outputs from these inputs if the system operates according to its specification.

Importance of Software Testing

- Provide confidence in the system Identify areas of weakness
- Establish the degree of quality
- Establish the extent that the requirements have been met
- To provide an understanding of the overall system
- To prove that the software is both usable and operable

Software Testability Checklist

- **Operability:** the better it works the more efficiently it can be tested
- **Observability:** what you see is what you test
- **Controllability:** the better software can be controlled the more testing can be automated and optimized
- **Decomposability:** by controlling the scope of testing, the more quickly problems can be isolated and retested intelligently
- **Simplicity:** the less there is to test, the more quickly we can test
- **Stability:** the fewer the changes, the fewer the disruptions to testing
- **Understandability:** the more information known, the smarter the testing

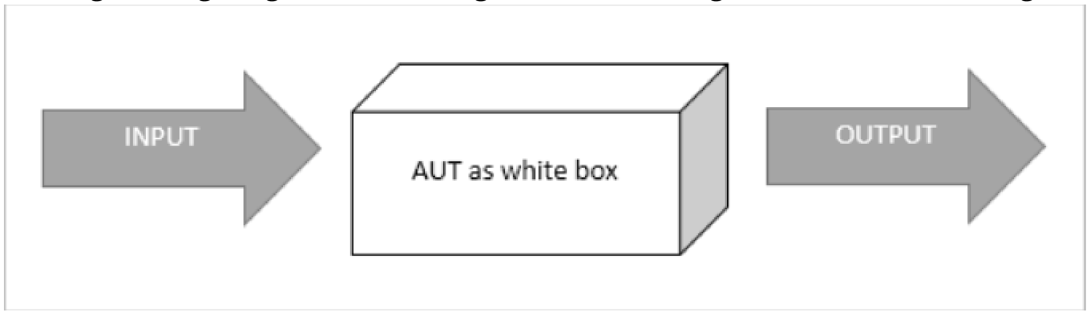
Benefits of Software Testing

- **Cost-Effective:** It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.
- **Security:** It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.
- **Product quality:** It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.
- **Customer Satisfaction:** The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

Techniques of Testing:

White-box testing

- Whitebox Testing is a software testing method in which the internal structure of the item being tested is known to the tester.
- White box testing is often used for verification.
- White box testing is done by the Developers.
- Also called Open testing, Clear-box testing, Code-based testing, Glass-box testing, Logic-coverage testing, Logic-driven testing, Structural testing, Structure-based testing, etc.



Example: A Car mechanic should know the internal structure of the car engine to repair it.
In this example,

- **CAR** is the **AUT (Application under Test).**
- **User** is the **black box tester.**
- **Mechanic** is the **white box tester.**

Advantage of Whitebox Testing

- It can be commenced at an earlier stage. One need not wait for the GUI to be available.
- It is more thorough, with the possibility of covering most paths.

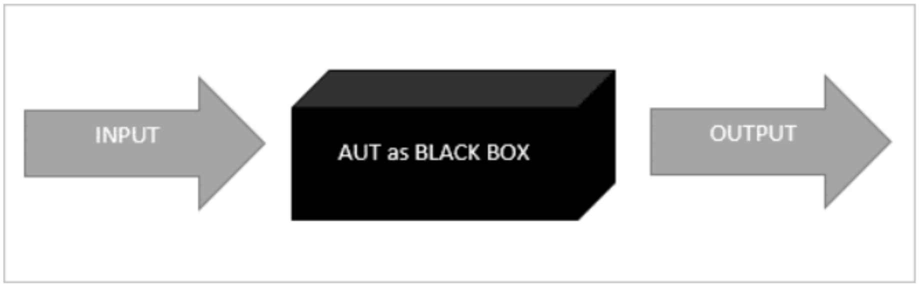
- As the testers of white box testing will have a programming background, it will be easy to identify the logical errors and inappropriate code segments.
- It helps in optimizing the code

Disadvantage of Whitebox Testing

- As knowledge of code and internal structure is a prerequisite, a skilled tester is needed to carry out this type of testing, which increases the cost
- Not looking at the code in a runtime environment.
- There is still scope for missing out on the hidden errors in the logic implemented.

Blackbox Testing

- Blackbox Testing is a software testing method in which the internal structure of the item being tested is not known to the tester.
    - In it, testing will be done by visualizing the application as a black box.
- Blackbox testing is often used for Validation.
- Black box testing is done by the professional testing team.
- Also called Specification-Based Testing, Closed Testing.



Example:
A simple example of black-box testing is a TV (Television). As a user, we watch the TV but we don’t need the knowledge of how the TV is built and how it works, etc. We just need to know how to operate the remote control to switch on, switch off, change channels, increase/decrease volume, etc.
In this example,

- The **TV** is your **AUT (Application under Test).**
- The **remote control** is the User Interface (UI) that you use to test.
- You just need to know how to use the application.

Advantage of Blackbox Testing

- Tester can be non-technical.
- Tester needs no knowledge of implementation, including specific programming languages
- Tests will be done from an end user's point of view. Because end user should accept the system.
- The design of test cases is easy.
- Black box testing can be automated very easily.

Disadvantage of Blackbox Testing

- Not all properties of a software product can be tested
- The reason for a failure is not found.
- Only a small number of possible inputs can be tested and many program paths will be left untested.

Levels of Testing

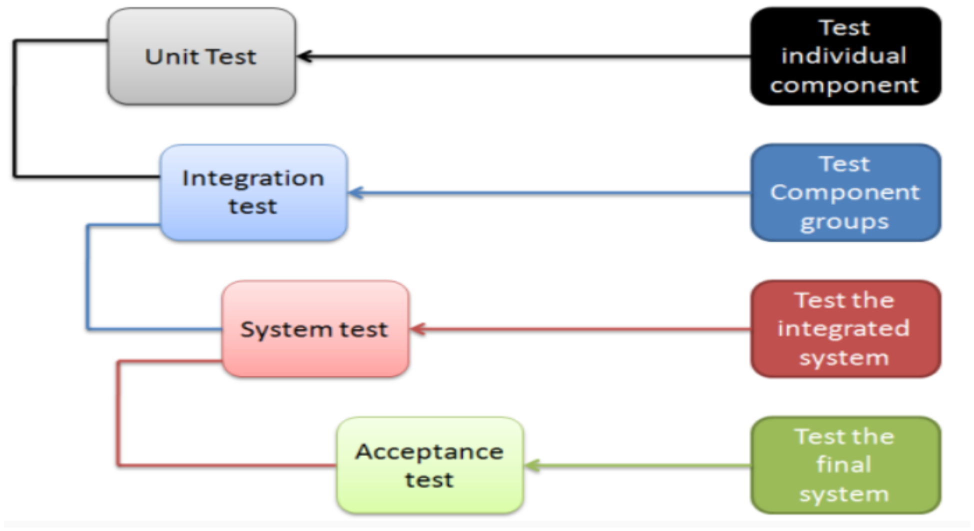


Fig: Overview of Software Testing Levels

1. Unit Testing:

- Unit testing is a method by which individual units of source code together are tested to determine if they are fit for use.
- The smallest independent and testable part of the source code is referred to as a unit. It could be a function, a file or a program.
- Unit tests are typically written and run by software developers to ensure that code meets its design and behaves as intended.

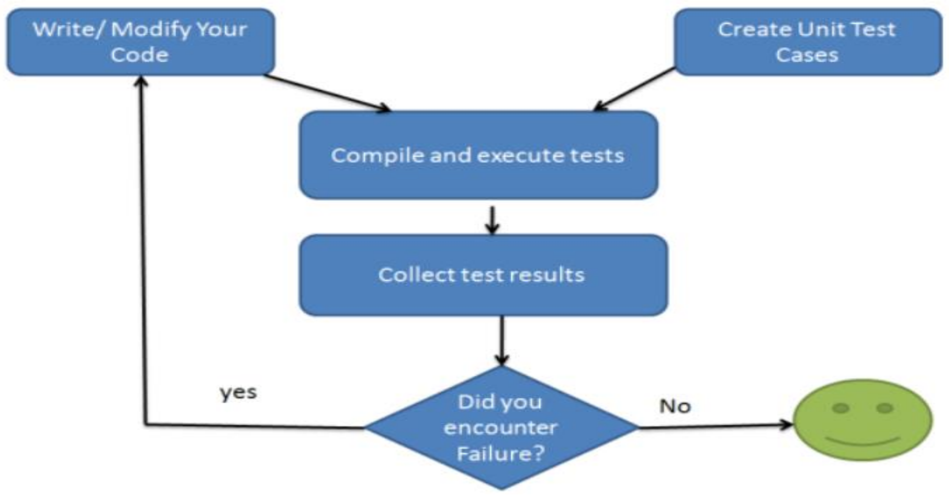


Fig: Unit Testing

2. Integration Testing:

- It is done after unit testing.
- In it, all the units are integrated together and tested as a group to make sure that integrated system is ready for system testing.
- It checks the data flow from one module to other modules. This kind of testing is performed by testers.

Types of Integration Testing

i. **Big bang testing:**

- As the name suggests, big bang form of testing all the modules are combined to form a complete system and then tested for bugs.

ii. **Top down testing:**

- It is a method in which integration testing takes place from top to bottom following the control flow of software system.
- The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality.

iii. **Bottom up testing:**

- It is a method in which the lower level modules are tested first.
- These tested modules are then further used to facilitate the testing of higher level modules. The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.

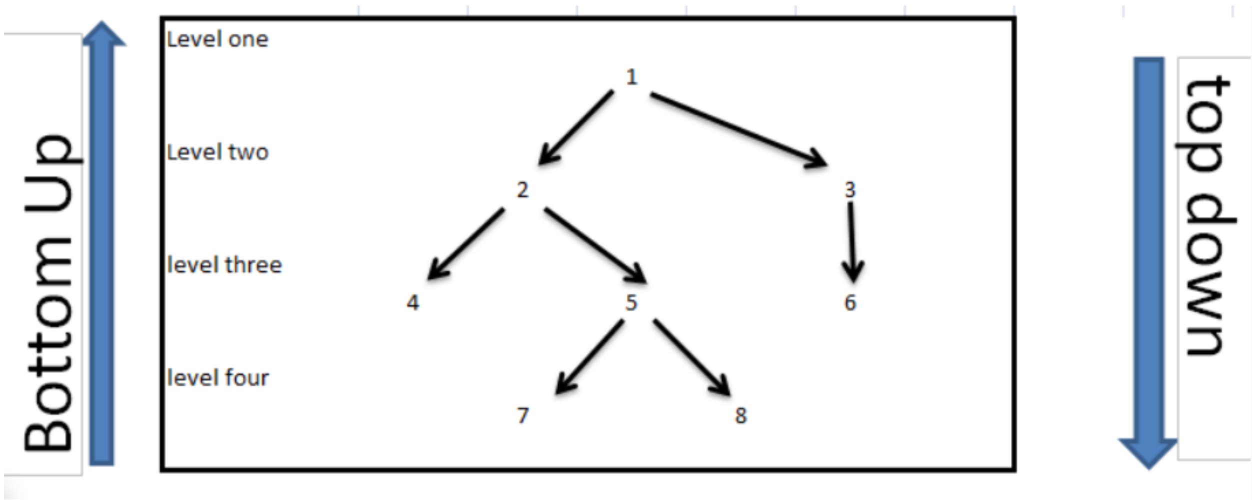


Fig: Top down and Bottom up Integration testing

3. System Testing:

- It is a level of software testing where a complete, integrated system is tested.
- The purpose of this test is to evaluate system's compliance with the specified requirements.
- After integration testing, fully integrated application is tested to check that where the system meets its software requirements specification (SRS).
- Falls under black box testing

- It is carried out to check the behavior of the application, software design and expectation of the end user.

4. Acceptance Testing:

- It is a test conducted to find if the requirements of a specification or contract are met as per its delivery.
- It is basically done by the user or customer. However, other stockholders can be involved in this process.

Types of Acceptance Testing:

i. Alpha Testing:

- It is a type of acceptance testing; performed to identify all possible issues and bugs before releasing the final product to the end users.
- Alpha testing is carried out by the testers who are internal employees of the organization.
- The main goal is to identify the tasks that a typical user might perform and test them.

Advantages of Alpha Testing:

- Provides better view about the reliability of the software at an early stage
- Helps simulate real time user behavior and environment.
- Detect many showstopper or serious errors
- Ability to provide early detection of errors with respect to design and functionality

Disadvantages of Alpha Testing:

- In depth, functionality cannot be tested as software is still under development stage. Sometimes developers and testers are dissatisfied with the results of alpha testing.

ii. Beta Testing:

- **It** is performed by “real users” of the software application in “real environment” and it can be considered as a form of external User Acceptance Testing.
- It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to test products in customer’s environment.
- Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality. Beta testing reduces product failure risks and provides increased quality of the product through customer validation.

Advantages of Beta Testing

- Reduces product failure risk via customer validation.
- Beta Testing allows a company to test post-launch infrastructure.
- Improves product quality via customer feedback
- Cost effective compared to similar data gathering methods
- Creates goodwill with customers and increases customer satisfaction

Disadvantages of Beta Testing

- Test Management is an issue. As compared to other testing types which are usually executed inside a company in a controlled environment, beta testing is executed out in the real world where you seldom have control.
- Finding the right beta users and maintaining their participation could be a challenge

Alpha Testing	Beta Testing
i. It is always done by developers at the software development site.	i. It is always performed by customers at their own site.
ii. It involves the use of both white box and black box testing techniques.	ii. It incorporates the use of black box technique in testing.
iii. It is performed in a simulated or virtual environment at the developer's site.	iii. It is executed in real time environment.
iv. Alpha testing is done in the absence of the targeted end user base.	iv. Beta testing is performed in the absence of the development/QA team.
v. It is performed at a stage when the user requirements are required.	v. It is also performed at a time when the software product is required to be marketed to the targeted clients/users.
vi. It is not open to the market and public.	vi. It is open to market and public.
vii. It is not useful in incorporating user feedback and issues.	vii. It incorporates user feedback towards improving the product quality.
viii. It is used for software applications and projects.	viii. It is used for software products.

Conventional Vs OO Systems

- **The structural differences:**
  - There are only functions, modules and subsystems in a conventional program whereas there are function members defined in a class, classes, groups of classes and subsystems in OO systems.
- **The behavior of the programs:**
  - There is a defined control flow among the active processes in a conventional program. But in an OO program, there are several active processes running on multiple processes to complete a function.
- **Dependencies:**
  - In conventional systems, there are data dependencies between variables, calling dependencies between modules, functional dependencies between a module and the variables it computes, definitional dependencies between a variable and its type; on the other hand, OO systems have additional dependencies such as class to class dependencies, class to method dependencies

Object Oriented Testing:

- It is a collection of testing techniques to verify and validate object-oriented software.
- Whenever large scale systems are designed, object oriented testing is done rather than conventional testing strategies as the concept of object oriented programming is way different from that of conventional ones.
- The whole object oriented testing revolves around the fundamental entity called “class”.
- With the help of “class” concept, larger system can be divided into small well defined units which may then be implemented separately.

- The object oriented testing can be classified as like conventional systems. These are called levels for testing.
- When should testing begin?
  - Analysis and Design
  - Programming
- To complete the OOT cycle mention below testing are required
  - Requirement Testing
  - Analysis and Design Testing
  - Code Testing
  - Integration Tests
  - System Tests
  - User Testing

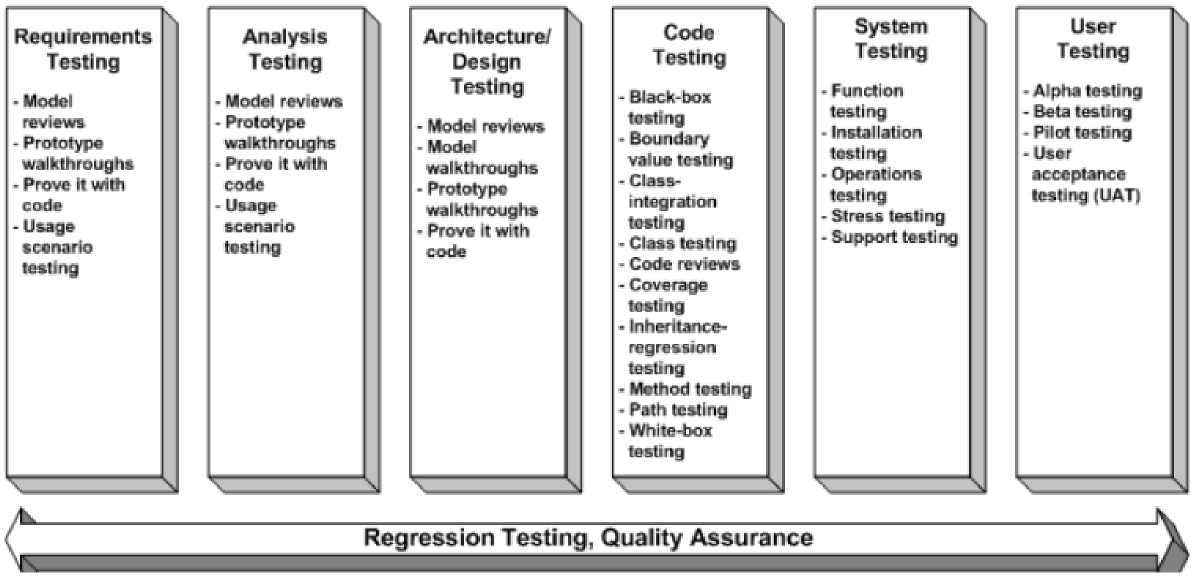


Fig: Full Life Cycle Object-Oriented Testing (FLOOT)

Testing OOA and OOD Models

- OO analysis and design models is especially useful because the same semantic constructs (e.g., classes, attributes, operations, messages) appear at the analysis, design, and code level.
- Analysis and design models cannot be tested in the conventional sense, because they cannot be executed.
- Formal technical review can be used to examine the correctness and consistency of both analysis and design models.
  - **Correctness:**
    - ✓ Syntax: Each model is reviewed to ensure that proper modeling conventions have been maintained.

    - ✓ Semantic: Must be judged based on the model's conformance to the real world problem domain by domain experts.
  - **Consistency:**
    - ✓ May be judged by considering the relationship among entities in the model.
    - ✓ Each class and its connections to other classes should be examined.
    - ✓ The Class-responsibility-collaboration model can be used.
  - **Completeness**

Object Oriented Testing Strategies:

1. Class Testing:

- Class testing is also known as unit testing.
- In class testing, every individual classes are tested for errors or bugs.
- Class testing ensures that the attributes of class are implemented as per the design and specifications. Also, it checks whether the interfaces and methods are error free of not.
- Unit testing is the responsibility of the application engineer who implements the structure.
- It involves three aspects:
  - Testing each method
  - Testing the relations among class methods
  - Testing the inheriting relation between class and subclass
- Rather than testing an individual module, the smallest testable unit is the encapsulated class or object. This is so because a class can contain a number of different operations and a particular operation may exist as part of a number of different classes.
- Class testing for OO software is the equivalent of unit testing for conventional ware.
- Unlike unit testing class testing for OO software is driven by the operations encapsulated by the class and the state behavior of the class.
- How is class testing different from conventional testing?
- Conventional testing focuses on input- process-output, whereas class testing focuses on each method, then designing sequences of methods to exercise states of a class.

Testing methods applicable at the class level

- Testing "in the small" focuses on a single class and the methods that are encapsulated by the class.
- Random testing and partitioning are methods that can be used to exercise a class during OO testing.

Random Testing for OO Classes:

- It requires large numbers data permutations and combinations, and can be inefficient
- Identify operations applicable to a class
- Define constraints on their use
- Identify a minimum test sequence
- Generate a variety of random test sequences.

Partition Testing at the Class Level:

- Partition testing reduces the number of test cases required to exercise the class in much the same manner as equivalence partitioning for traditional software.
- Input and output are categorized and test cases are designed to exercise each category.

- **State-based partitioning:**
  - It categorizes class operations based on their ability to change the state of the class.
  - Test designed in such a way that operations that cause state changes are tested separately from those that do not.
- **Attribute-based partitioning:**
  - For each class attribute, operations are classified according to those that use the attribute, those that modify the attribute and those that do not use or modify the attributes.
- **Category-based partitioning:**
  - It categorizes class operations based on the generic function that each performs.
  - For example, operations in the Account class can be categorized in initialization operations (open, setup), computational operations (deposit, withdraw), queries (balance, summarize, creditLimit), and termination operations (close).

2. Integration testing:

- It is also called as interclass or subsystem testing.
- Inter class testing involves the testing of modules or sub-systems and their coordination with other modules.

Different strategies for integration testing of OO systems:

a. **Thread-based testing:**
   - Integrates classes required to respond to one input or event.
   - Thread-based testing integrates the set of classes required to respond to one input or event for the system. Each thread is integrated and tested individually.

b. **Use-based testing:**
   - It begins the construction of the system by testing those classes (called independent classes) that use very few (if any) of server classes.
   - After the Independent classes are tested, the next layer of classes, called dependent classes, that use the independent classes are tested.
   - This sequence of testing layers of dependent classes continue until the entire system is constructed.

c. **Cluster testing:**
   - A cluster of collaborating classes (determined by examine the CRC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

3. Validation Testing:

- It focuses on user-visible actions and user-recognizable output from the system.
- To assist in the derivation of validation testing, the tester should work upon the use-cases that are part of the analysis model.
- The use-case provides a scenario that has a high likelihood of uncovered errors in the user interaction requirements.
- Conventional black-box testing methods can be used to drive validation tests.
- In addition, test cases may be derived from the object-behavior model and from event flow diagram created as part of OOA.

4. System Testing:

- Software may be part of a larger system. This often leads to "finger pointing" by other system dev teams.
- Finger pointing defence:
    1. Design error-handling paths that test external information
    2. Conduct a series of tests that simulate bad data
    3. Record the results of tests to use as evidence
- Types of System Testing:

i. **Recovery testing**: how well and quickly does the system recover from faults

ii. **Security testing**: verify that protection mechanisms built into the system will protect from unauthorized access (hackers, disgruntled employees, fraudsters)

iii. **Stress testing**: place abnormal load on the system

iv. **Performance testing**: investigate the run-time performance within the context of an integrated system

Issues in OO Methods Testing

Traditional testing methods are not directly applicable to OO programs as they involve OO concepts including polymorphism, inheritance and polymorphism. These concepts leads to issues which are yet to be resolved. Some of the issues are:

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).

Encapsulation Issues:

- Encapsulation requires that classes are only aware of their own properties, and are able to operate independently.
- 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.
- If you violate encapsulation for testing purposes, then the validity of test could be questionable.

Inheritance Issues:

- Inheritance is an important part of the object oriented paradigm.
- It introduce problems that are not found in traditional software.
- OO 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.

Implication of Genericity

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

Polymorphism Issues:

- Each possible binding of polymorphic component requires a separate 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.

Implications for testing processes

- Re-examine all testing techniques and processes.

Object-oriented testing methods

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.

Scenario-Based Test Design:

- Fault-based testing misses two main types of errors:
  - incorrect specifications and
  - Interactions among subsystems.
- When errors associated with an incorrect specification occur, the product doesn't do what the customer wants. It might do the wrong thing or omit important functionality.
- But in either circumstance, quality (conformance to requirements) suffers. Errors associated with subsystem interaction occur when the behavior of one subsystem creates circumstances (e.g., events, data flow) that cause another subsystem to fail.
- Scenario-based testing concentrates on what the user does, not what the product does.
- This means capturing the tasks (via use cases) that the user has to perform and then applying them and their variants as tests. Scenarios uncover interaction errors.
- But to accomplish this, test cases must be more complex and more realistic than fault-based tests. Scenario-based testing tends to exercise multiple subsystems in a single test (users do not limit themselves to the use of one subsystem at a time.

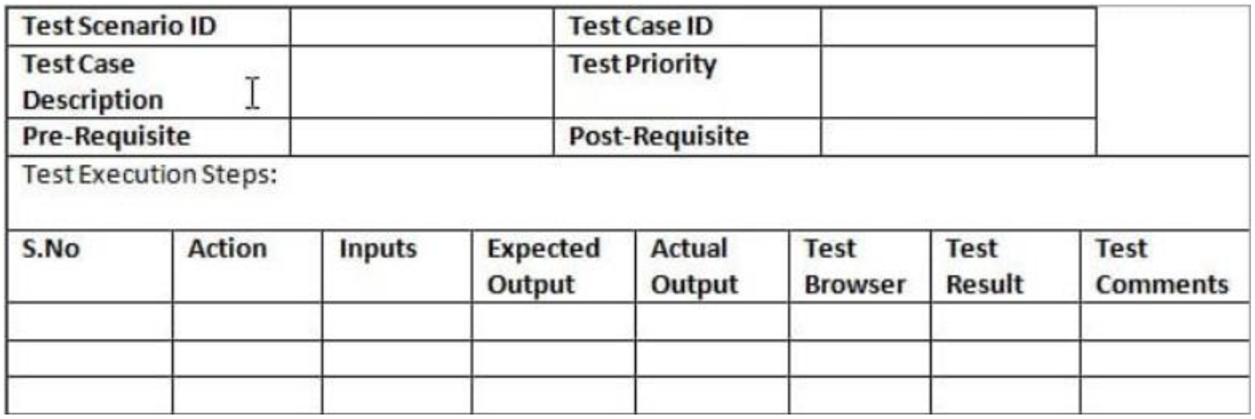
Develop Test Cases and Test Plans

Test Case:

- It is a set of actions executed to verify a particular feature or functionality of your software application.
- A test case is a specification of the inputs, execution conditions, testing procedure, and expected results that define a single test to be executed to achieve a particular software testing objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
- A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

Test Case Design for OO software

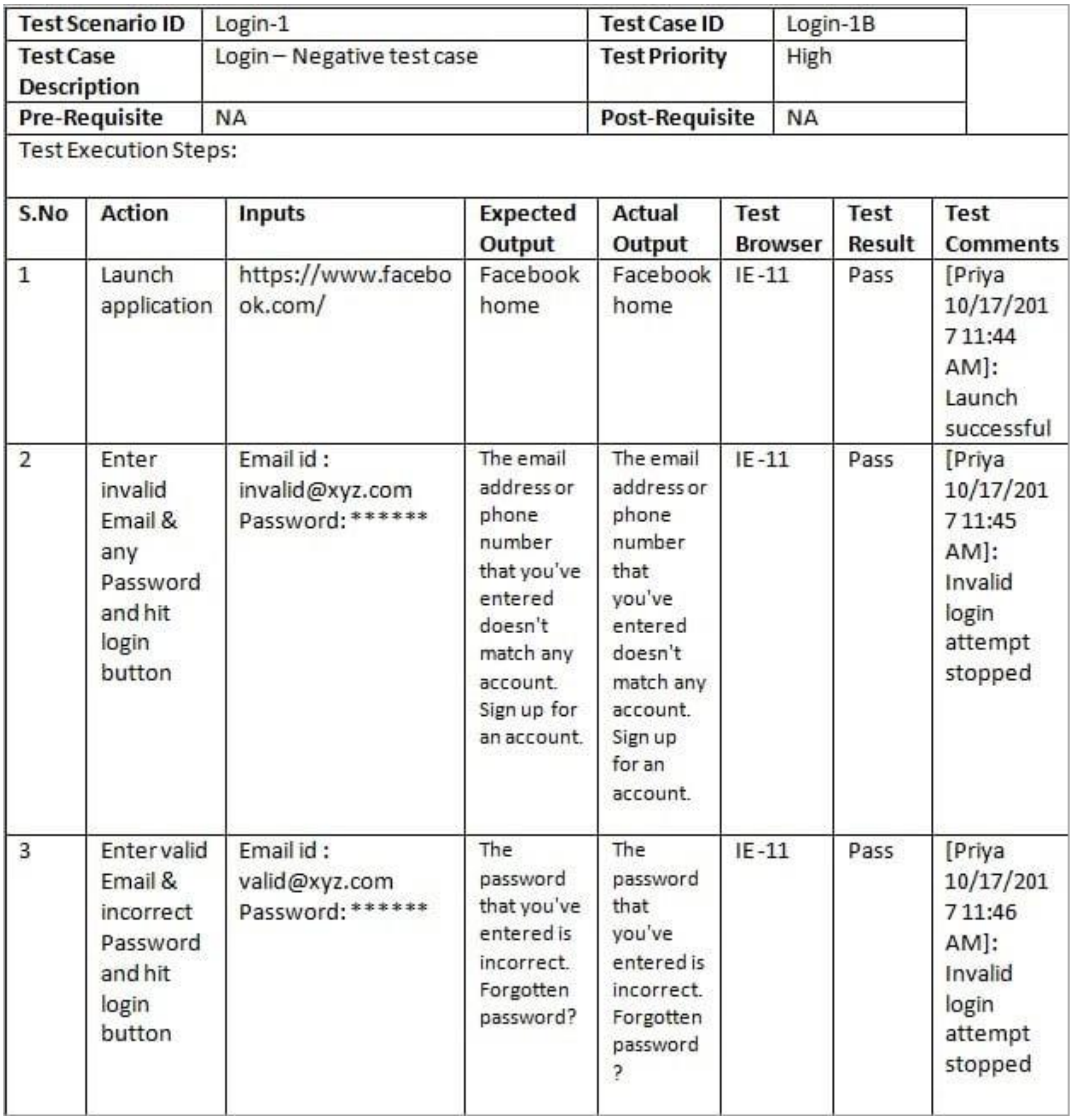
- An overall approach to OO test case design has been defined by Berard:
- Each test case should be uniquely identified and explicitly associated with the class to be tested.
- The purpose of the test should be stated.
- A list of testing steps should be developed for each test and should contain:
  - A list of specified states for the object that is to be tested.
  - A list of messages and operations that will be executed as a consequence of the test.
  - A list of exceptions that may occur as the object is being tested.
  - A list of external conditions (i.e., the changes in the environment external to the software that must exist in order to properly conduct the test.)
  - Supplementary information that will aid in understanding or implementing the test.
- Unlike conventional test case design, which is driven by an input-process-output view of software or the algorithmic detail of individual modules.
- Object-oriented testing focuses on designing appropriate sequences of operations to exercise the states of a class.



Test Scenario ID		Test Case ID		
Test Case Description		Test Priority		
Pre-Requisite		Post-Requisite		
Test Execution Steps:				

S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result	Test Comments

Fig: Sample Test Case

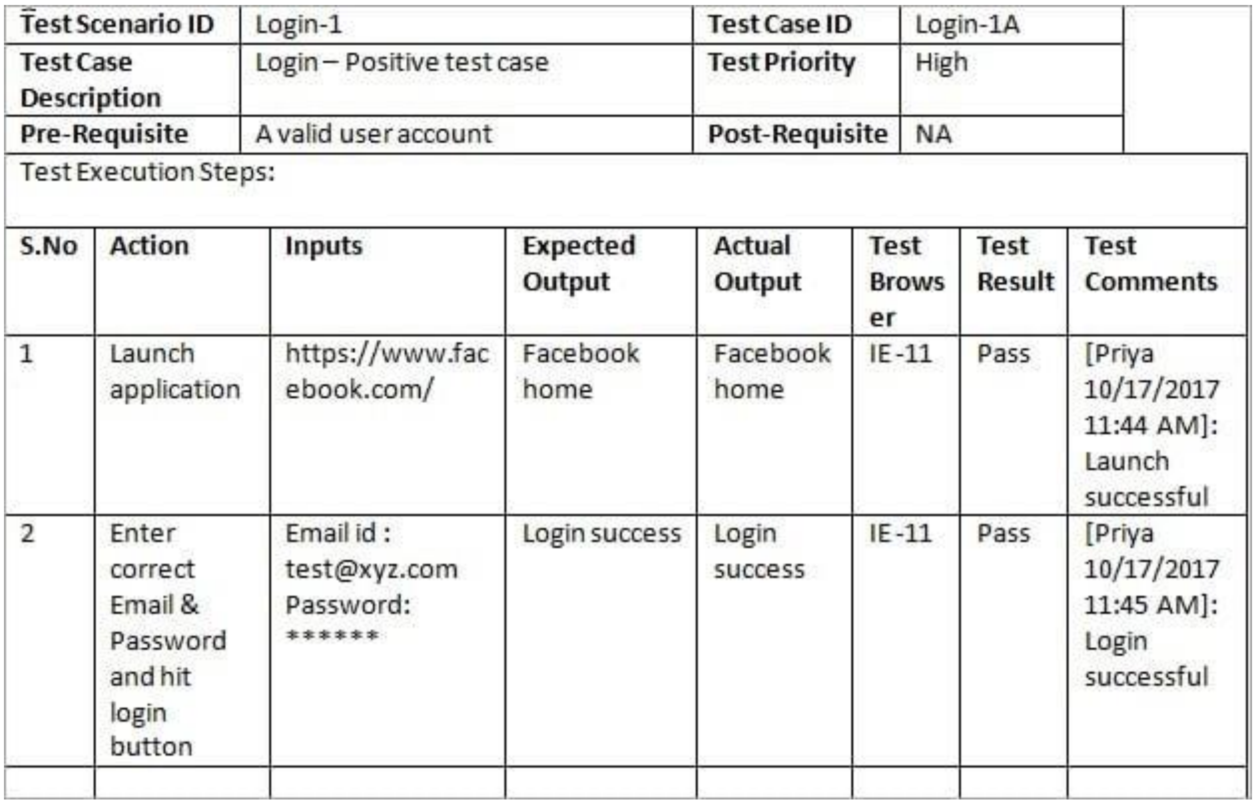


Test Scenario ID	Login-1	**Test Case ID**	Login-1B
Test Case Description	Login – Negative test case	**Test Priority**	High
Pre-Requisite	NA	**Post-Requisite**	NA

Test Execution Steps:

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**	**Test Comments**
1	Launch application	https://www.facebook.com/	Facebook home	Facebook home	IE-11	Pass	[Priya 10/17/2017 11:44 AM]: Launch successful
2	Enter invalid Email & any Password and hit login button	Email id : invalid@xyz.com Password: ******	The email address or phone number that you've entered doesn't match any account. Sign up for an account.	The email address or phone number that you've entered doesn't match any account. Sign up for an account.	IE-11	Pass	[Priya 10/17/2017 11:45 AM]: Invalid login attempt stopped
3	Enter valid Email & incorrect Password and hit login button	Email id : valid@xyz.com Password: ******	The password that you've entered is incorrect. Forgotten password?	The password that you've entered is incorrect. Forgotten password?	IE-11	Pass	[Priya 10/17/2017 11:46 AM]: Invalid login attempt stopped

Fig: Negative Test Cases for Login



Test Scenario ID	Login-1	**Test Case ID**	Login-1A
Test Case Description	Login – Positive test case	**Test Priority**	High
Pre-Requisite	A valid user account	**Post-Requisite**	NA

Test Execution Steps:

S.No	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Browser**	**Test Result**	**Test Comments**
1	Launch application	https://www.facebook.com/	Facebook home	Facebook home	IE-11	Pass	[Priya 10/17/2017 11:44 AM]: Launch successful
2	Enter correct Email & Password and hit login button	Email id : test@xyz.com Password: ******	Login success	Login success	IE-11	Pass	[Priya 10/17/2017 11:45 AM]: Login successful

Fig: Positive Test Cases for Login

Test Case Examples

Test case for ATM

TC 1:- successful card insertion.
TC 2:- unsuccessful operation due to wrong angle card insertion.
TC 3:- unsuccessful operation due to invalid account card.
TC 4:- successful entry of pin number.
TC 5:- unsuccessful operation due to wrong pin number entered 3 times.
TC 6:- successful selection of language.
TC 7:- successful selection of account type.
TC 8:- unsuccessful operation due to wrong account type selected w/r to that inserted card.
TC 9:- successful selection of withdrawal option.
TC 10:- successful selection of amount.
TC 11:- unsuccessful operation due to wrong denominations.
TC 12:- successful withdrawal operation.
TC 13:- unsuccessful withdrawal operation due to amount greater than possible balance.
TC 14:- unsuccessful due to lack of amount in ATM.
TC 15:- unsuccessful due to amount greater than the day limit.
TC 16:- unsuccessful due to server down.
TC 17:- unsuccessful due to click cancel after insert card.
TC 18:- unsuccessful due to click cancel after insert card and pin no.
TC 19:-unsuccessful due to click cancel after language selection, account type selection, withdrawal selection, enter amount.

Test cases for a web page

- Testing without entering any username and password
- Test it only with Username Test it only with password.
- User name with wrong password
- Password with wrong user name
- Right username and right password
- Cancel, after entering username and password.
- Enter long username and password that exceeds the set limit of characters.
- Try copy/paste in the password text box.
- After successful sign-out, try "Back" option from your browser.
- Check whether it gets you to the "signed-in" page.

Interclass test-case design

- Test-case design becomes more complicated as integration of the object-oriented system begins. It is at this stage that testing of collaborations between classes must begin.
- Like the testing of individual classes, class collaboration testing can be accomplished by applying random and partitioning methods, as well as scenario-based testing and behavioral testing.

Multiple Class Testing:

- For each client class, use the list of class operations to generate a series of random test sequences. The operations will send messages to other server classes.
- For each message that is generated, determine the collaborator class and the corresponding operation in the server object.
- For each operation in the server object (that has been invoked by messages sent from the client object), determine the messages that it transmits.
- For each of the messages, determine the next level of operations that are invoked and incorporate these into the test sequence.

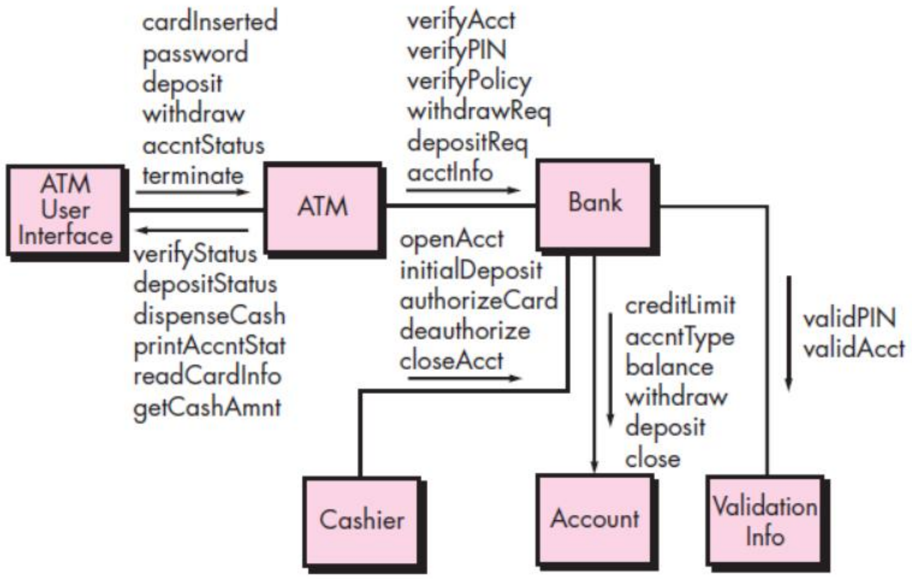


Fig: Class Collaboration Diagram for banking application

CRC Cards

- **Class Responsibility Collaboration (CRC**) cards
- A CRC card is a physical card representing a single class
- Each card lists the class's name, attributes and methods (its responsibilities), and class associations (collaborations).
- The collection of these CRC cards is the CRC model.
- **Steps for CRC Cards techniques:**
  - Identify the classes.
  - List responsibilities.
  - List collaborators.

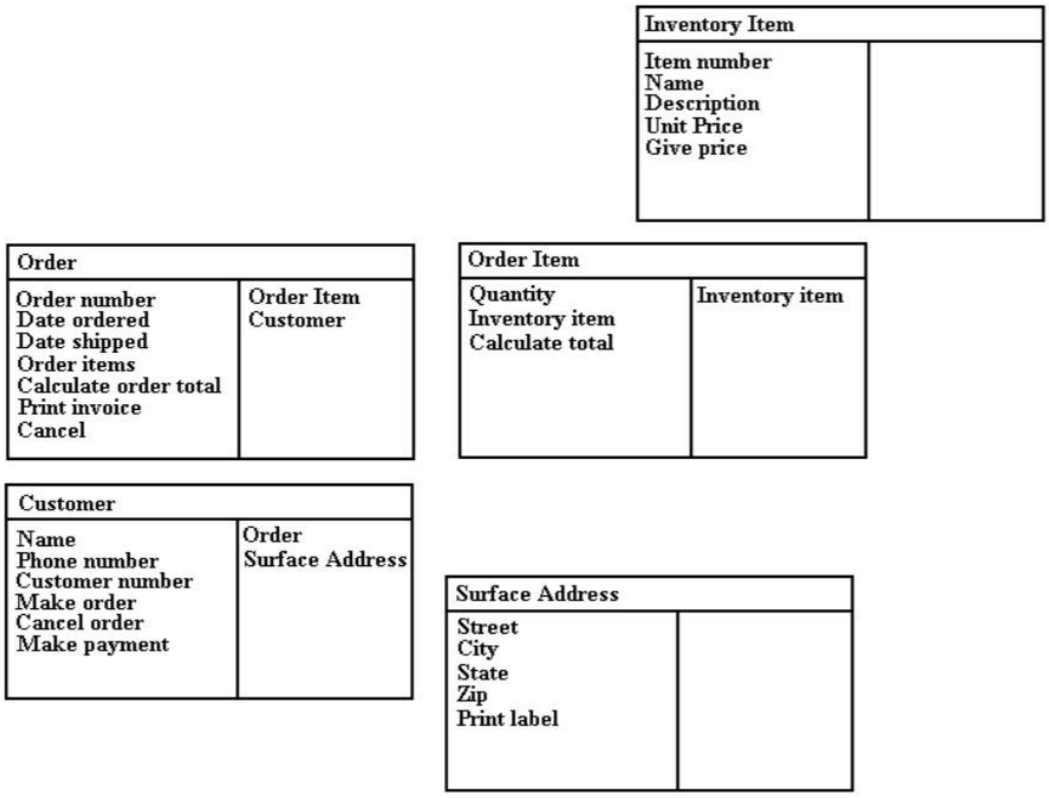
CRC Cards

- Class: A Class represents a collection of similar objects. The Class name appears across the top of the CRC card.
- Responsibility: A Responsibility is anything that the class knows or does. It appear along the left side of the CRC card.
- Collaborator: A Collaborator is another class that is used to get information for, or perform actions for the class at hand. It often works with a particular class to complete a step (or steps) in a scenario. The Collaborators of a class appear along the right side of the CRC card

CRC Cards

Class Name	
Responsibilities	Collaborators

Student	
Student number Name Address Phone number Enroll in a seminar Drop a seminar Request transcripts	Seminar



Inventory Item	
Item number Name Description Unit Price Give price	

Order	
Order number Date ordered Date shipped Order items Calculate order total Print invoice Cancel	Order Item Customer

Order Item	
Quantity Inventory item Calculate total	Inventory item

Customer	
Name Phone number Customer number Make order Cancel order Make payment	Order Surface Address

Surface Address	
Street City State Zip Print label	

Figure 2. A CRC model for a simple shipping/inventory control system.