the **database** and at the **business level** it appears as **data warehouse and data mining**. Thus data plays an important role in software design.

6.5.2 Architectural Design Element

The architectural design gives the layout for overall view of the software. Architectural model can be built using following sources -

- Data flow models or class diagrams.
- Information obtained from application domain.
- Architectural patterns and styles.

6.5.3 Interface Design Elements

Interface Design represents the **detailed design** of the software system. In interface design how **information flows** from one component to other component of the system is depicted. Typically there are three types of interfaces -

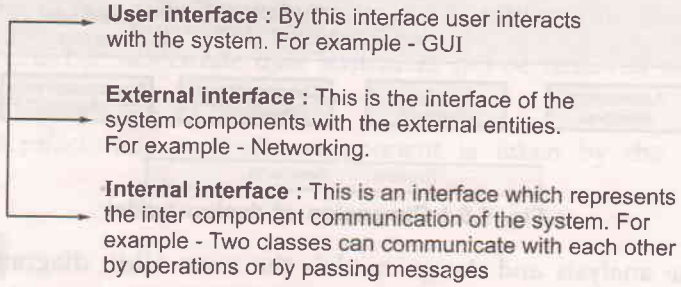

Fig. 6.5.2 Interface design elements

6.5.4 Component Level Design Elements

The component level design is more detailed design of the software system along with the specifications. The component level design elements describe the internal details of the component. In component level design all the local data objects, required data structures and algorithmic details and procedural details are exposed.

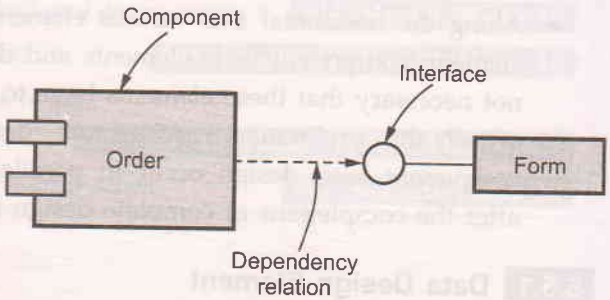

Fig. 6.5.3 Components

Fig. 6.5.3 represents that component **order** makes use of another component **form**.

The **order** is dependent upon the component form. These two objects can be interfaced with each other.

6.5.5 Deployment Level Design Elements

The deployment level design elements indicate how software functions and software subsystems are assigned to the physical computing environment of the software product. For example web browsers may work in mobile phoned or they may run on client PC or can execute on server machines.

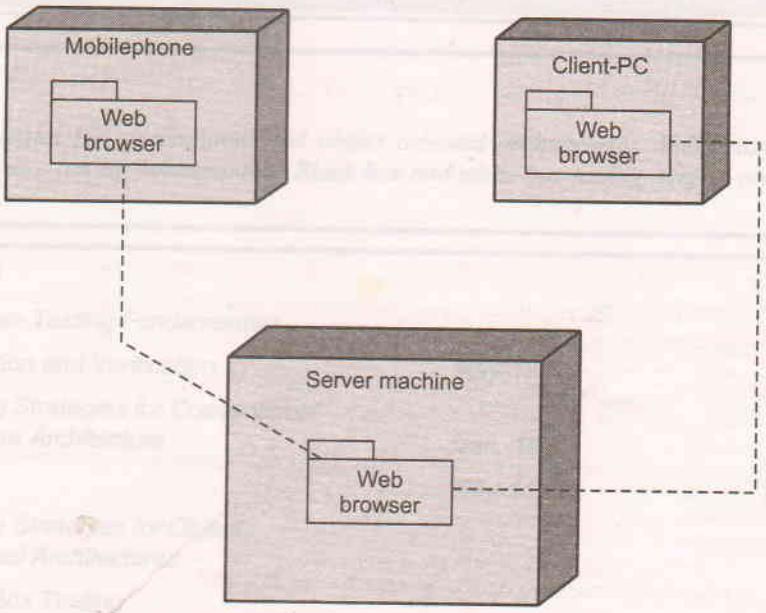

Fig. 6.5.4 Deployment diagram

❑❑❑

7.1 Software Testing Fundamentals

*Definition : **Software testing** is a critical element of software quality assurance and represents the ultimate review of specification, design and coding.*

The purpose of software testing is to ensure whether the software functions appear to be working according to specifications and performance requirements.

7.1.1 Testing Objectives

According to **Glen Myers** the testing objectives are

1. Testing is a process of executing a program with the intend of finding an error.
2. A good test case is one that has high probability of finding an undiscovered error.
3. A successful test is one that uncovers an as-yet undiscovered error.

The major testing objective is to design tests that systematically uncover types of errors with minimum time and effort.

7.1.2 Testing Principles

Every software engineer must apply following testing principles while performing the software testing.

1. All tests should be traceable to customer requirements.
2. Tests should be planned long before testing begins.
3. The Pareto principle can be applied to software testing - 80 % of all errors uncovered during testing will likely be traceable to 20 % of all program modules.
4. Testing should begin "in the small" and progress toward testing "in the large".
5. Exhaustive testing is not possible.
6. To be most effective, testing should be conducted by an independent third party.

7.1.3 Why Testing is Important ?

- Generally, testing is a process that requires more efforts than any other software engineering activity.
- Testing is a set of activities that can be planned in advance and conducted systematically.
- If it is conducted haphazardly, then only time will be wasted and more even worse errors may get introduced.
- This may lead to have many undetected errors in the system being developed. Hence performing testing by adopting systematic strategies is very much essential in during development of software.

Review Question

1. *What is importance of testing practices ? What are the principles of testing practices ?*

7.2 Validation and Verification

MU : May-13, Marks 10

- Verification refers to the set of activities that ensure that software correctly implements a specific function.
- Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements.
- According to Boehm

  Verification : "Are we building the product right ?"

  Validation : "Are we building the right product ?"
- Software testing is only one element of Software Quality Assurance (SQA).
- Quality must be built into the development process, you can't use testing to add quality after the fact.
- Verification and validation involve large number of software quality assurance activities such as -
  - Formal technical reviews
  - Quality and configuration audits
  - Performance monitoring
  - Feasibility study
  - Documentation review
  - Database review
  - Algorithmic analysis
  - Development testing
  - Installation testing

Comparison of Verification and Validation

Verification	Validation
Verification refers to the set of activities that ensure software **correctly implements** the specific function.	**Validation** refers to the set of activities that ensure that the software that has been built is **traceable to customer requirements.**
Are we building the product right ?	Are we building the right product ?
After a valid and complete specification the verification starts.	Validation begins as soon as project starts.
The verification is conducted to ensure whether software meets specification or not.	Validation is conducted to show that the user requirements are getting satisfied.

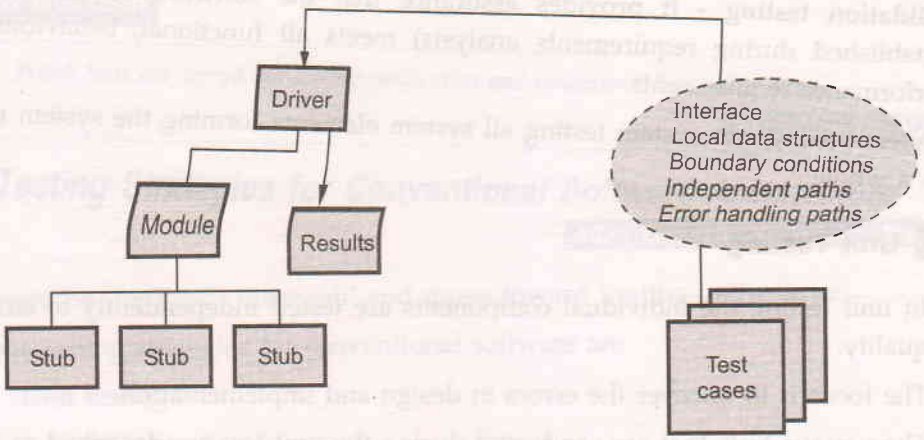

Fig. 7.3.3 Unit testing environment

7. The unit testing is simplified when a component with high cohesion (with one function) is designed. In such a design the number of test cases are less and one can easily predict or uncover errors.

7.3.2 Integration Testing

- A group of dependent components are tested together to ensure their quality of their integration unit.
- The objective is to take unit tested components and build a program structure that has been dictated by software design.
- The focus of integration testing is to uncover errors in :
  - Design and construction of software architecture.
  - Integrated functions or operations at subsystem level.
  - Interfaces and interactions between them.
  - Resource integration and/or environment integration.
- The integration testing can be carried out using two approaches.
  1. The non-incremental integration 2. Incremental integration
- The non-incremental integration is given by the "**big bang**" approach. All components are combined in advance. The entire program is tested as a whole. And chaos usually results. A set of errors is tested as a whole. Correction is difficult because isolation of causes is complicated by the size of the entire program. Once these errors are corrected new ones appear. This process continues infinitely.

Advantage of big-bang : This approach is simple.

Disadvantages : 1. It is hard to debug.

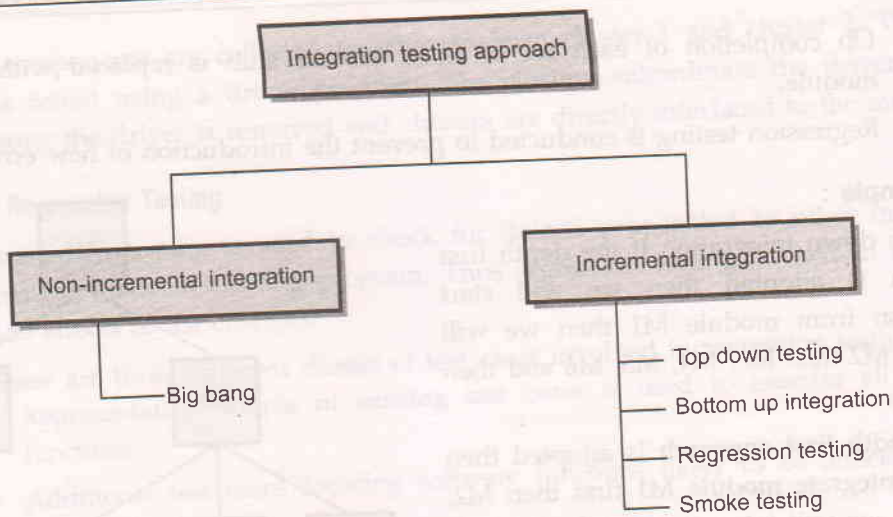

Fig. 7.3.4 Integration testing approach

2. It is not easy to isolate errors while testing.
3. In this approach it is not easy to validate test results.
4. After performing testing, it is impossible to form an integrated system.

- An incremental construction strategy includes
  Top down integration
  Bottom up integration
  Regression testing
  Smoke testing

7.3.2.1 Top Down Integration Testing

- Top down testing is an incremental approach in which modules are integrated by moving down through the control structure.
- Modules subordinate to the main control module are incorporated into the system in either a depth first or breadth first manner.
- Integration process can be performed using following steps.
  1. The main control module is used as a test driver and the stubs are substituted for all modules directly subordinate to the main control module.
  2. Subordinate stubs are replaced one at a time with actual modules using either depth first or breadth first method.
  3. Tests are conducted as each module is integrated.

4. On completion of each set of tests, another stub is replaced with the real module.
5. Regression testing is conducted to prevent the introduction of new errors.

For example :

In top down integration if the depth first approach is adopted then we will start integration from module M1 then we will integrate M2 then M3, M4, M5, M6 and then M7.

If breadth first approach is adopted then we will integrate module M1 first then M2, M6. Then we will integrate module M3, M4, M5 and finally M7.

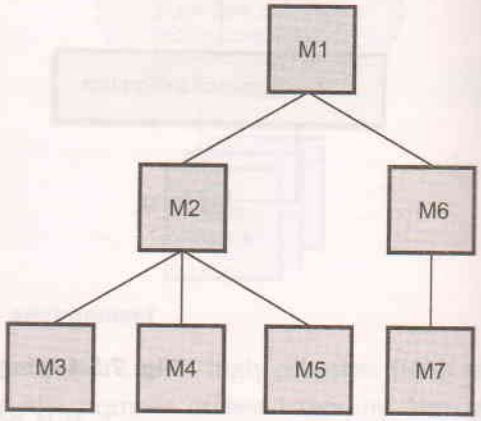

Fig. 7.3.5 Program structure

7.3.2.2 Bottom Up Integration Testing

In bottom up integration the modules at the lowest levels are integrated at first, then integration is done by moving upward through the control structure.

The bottom up integration process can be carried out using following steps.

1. Low-level modules are combined into clusters that perform a specific software subfunction.
2. A driver program is written to co-ordinate test case input and output.
3. The whole cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

For example :

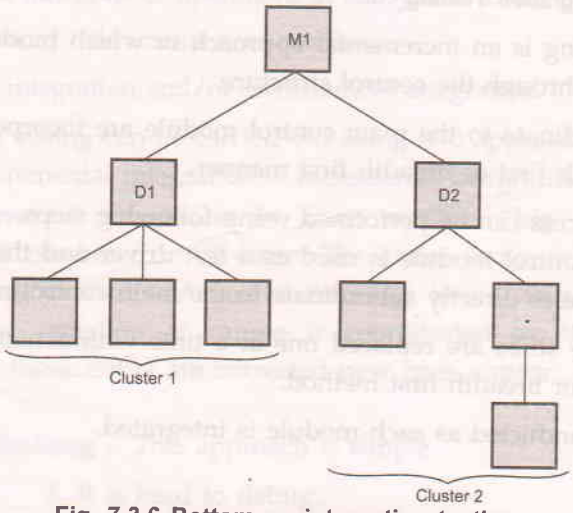

Fig. 7.3.6 Bottom up integration testing

First components are collected together to form cluster 1 and cluster 2. Then each cluster is tested using a driver program. The clusters subordinate the driver module. After testing the driver is removed and clusters are directly interfaced to the modules.

7.3.2.3 Regression Testing

- Regression testing is used to check for defects propagated to other modules by changes made to existing program. Thus regression testing is used to reduce the side effects of the changes.
- There are three different classes of test cases involved in regression testing -
  - Representative sample of existing test cases is used to exercise all software functions.
  - Additional test cases focusing software functions likely to be affected by the change.
  - Tests cases that focus on the changed software components.
- After product had been deployed, regression testing would be necessary because after a change has been made to the product an error that can be discovered and it should be corrected. Similarly for deployed product addition of new feature may be requested and implemented. For that reason regression testing is essential.

7.3.2.4 Smoke Testing

- The smoke testing is a kind of integration testing technique used for time critical projects wherein the project needs to be assessed on frequent basis.
- Following activities need to be carried out in smoke testing -
  1. Software components already translated into code are integrated into a "build". The "build" can be data files, libraries, reusable modules or program components.
  2. A series of tests are designed to expose errors from build so that the "build" performs its functioning correctly.
  3. The "build" is integrated with the other builds and the entire product is smoke tested daily.

Smoke testing benefits

1. Integration risk is minimized.
2. The quality of the end product is improved.
3. Error diagnosis and correction are simplified.
4. Assessment of progress is easy.

7.3.3 Validation Testing

- The integrated software is tested based on requirements to ensure that the desired product is obtained.
- In validation testing the main focus is to uncover errors in
  - System input/output
  - System functions and information data
  - System interfaces with external parts
  - User interfaces
  - System behaviour and performance
- Software validation can be performed through a series of **black box tests.**
- After performing the validation tests there exists two conditions.
  1. The function or performance characteristics are **according** to the **specifications** and are accepted.
  2. The requirement specifications are derived and the deficiency list is created. The **deficiencies** then can be **resolved** by establishing the proper communication with the customer.
- Finally in validation testing a review is taken to ensure that all the elements of software configuration are developed as per requirements. This review is called configuration review or audit.

7.3.3.1 Acceptance Testing

The acceptance testing is a kind of testing conducted to ensure that the software works correctly in the user work environment.

The acceptance testing can be conducted over a period of weeks or months.

The types of acceptance testing are

1. **Alpha test** - The alpha testing is a testing in which the version of complete software is tested by the customer under the supervision of developer. This testing is performed at developer's site. The software is used in natural setting in presence of developer. This test is conducted in controlled environment.
2. **Beta test** - The beta testing is a testing in which the version of software is tested by the customer without the developer being present. This testing is performed at customer's site. As there is no presence of developer during testing, it is not controlled by developer. The end user records the problems and report them to developer. The developer then makes appropriate modification.

Difference between Alpha and Beta Testing

Alpha testing	Beta testing
This testing is done by a developer or by a customer under the supervision of developer in company's premises.	This testing is done by the customer without any interference of developer and is done at customer's place.
Sometime full product is not tested using alpha testing and only core functionalities are tested.	The complete product is tested under this testing. Such product is usually given as free trial version.

7.3.4 System Testing

The system test is a series of tests conducted to fully the computer based system.

Various types of system tests are

1. **Recovery testing**
2. **Security testing**
3. **Stress testing**
4. **Performance testing**

The main focus of such testing is to test

- System functions and performance.
- System reliability and recoverability (recovery test).
- System installation (installation test).
- System behaviour in the special conditions (stress test).
- System user operations (acceptance test/alpha test).
- Hardware and software integration and collaboration.
- Integration of external software and the system.

7.3.4.1 Recovery Testing

- Recovery testing is intended to check the system's ability to recover from failures.
- In this type of testing the software is forced to fail and then it is verified whether the system recovers properly or not.
- For automated recovery then reinitialization, checkpoint mechanisms, data recovery and restart are verified.

7.3.4.2 Security Testing

- Security testing verifies that system protection mechanism prevent improper penetration or data alteration.

- It also verifies that protection mechanisms built into the system prevent intrusion such as unauthorized internal or external access or willful damage.
- System design goal is to make the penetration attempt more costly than the value of the information that will be obtained.

7.3.4.3 Stress Testing

- Determines breakpoint of a system to establish maximum service level.
- In stress testing the system is executed in a manner that demands resources in abnormal quantity, frequency or volume.
- A variation of stress testing is a technique called sensitivity testing.
- The sensitive testing is a testing in which it is tried to uncover data from a large class of valid data that may cause instability or improper processing.

7.3.4.4 Performance Testing

- Performance testing evaluates the run time performance of the software, especially real time software.
- In performance testing resource utilization such as CPU load, throughput, response time, memory usage can be measured.
- For big systems (e.g. banking systems) involving many users connecting to servers (e.g. using internet) performance testing is very difficult.
- Beta testing is useful for performance testing.

Review Questions

1. *Explain the validation testing type.*
2. *Differentiate between alpha testing and beta testing.*
3. *Write a note on system testing.* **MU : Dec.-10, Marks 10, May-11,12, Marks 5**

7.4 Testing Strategies for Object Oriented Architectures

- The primary objective of testing for object oriented software is to **uncover as much errors as** possible with manageable amount of efforts in realistic time span.
- The object oriented testing strategy is identical to conventional testing strategy. The strategy is start **testing in small** and work outward to **testing in large**. The basic unit of testing is **class** that contains attributes and operations. These classes **integrate** to form object oriented architecture. These are called **collaborating classes.**

7.4.1 Unit Testing in OO Context

- **Class** is an encapsulation of data attributes and corresponding set of operations.
- The **object** is an instance of a class. Hence objects also specify some data attributes and operations.
- In object oriented software the focus of unit testing is considered as classes or objects.
- However, operations within the classes are also the testable units. In fact in OO context the operation can not be tested as single isolated unit because one operation can be defined in one particular class and can be used in multiple classes at the same time. For example consider the operation **display()**. This operation is defined as super class an at the same time it can be used by the can be multiple derived classes. Hence it is not possible to test the operation as single module.
- Thus class testing for OO software is equivalent to unit testing for conventional software. In conventional software system, the algorithmic details and data that flow are units of testing but in OO software the encapsulated classes and operations (that are encapsulated within the class and state behavior of class) are the unit of testing.

7.4.2 Integration Testing in OO Context

- There are two strategies used for integration testing and those are -
  1. Thread based testing
  2. Use-based testing.
- The **thread based testing** integrates all the classes that respond to one input or event for the system. Each thread is then tested individually.
- In **use-based testing** the independent classes and dependent classes are tested. The **independent classes** are those classes that uses the server classes and **dependant classes** are those classes that use the independent classes. In use based testing the independent classes are tested at the beginning and the testing proceeds towards testing of dependent classes.

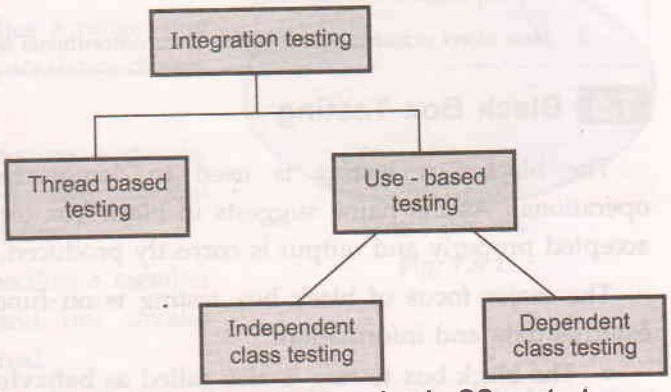

Fig. 7.4.1 Integration testing in OO context