Unit – 4
Testing and Implementation

Software testing fundamentals-Internal and external views of Testing-white box testing- basis path testing-control structure testing-black box testing- Regression Testing – Unit Testing – Integration Testing – Validation Testing – System Testing and Debugging – Software Implementation Techniques: Coding practices-Refactoring – Maintenance and Reengineering – BPR model – Reengineering process model – Reverse and Forward Engineering.

Software testing fundamentals

4.1 Definition of Testing

Software testing is a critical element of software quality assurance and f 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.

4.1.1 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.

4.1.2. 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.

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

Defects and failures

Not all software defects are caused by coding errors. One common source of expensive defects is requirement gaps, e.g., unrecognized requirements which result in errors of omission by the program designer. Requirement gaps can often be non-functional requirements such as testability, scalability, maintainability, usability, performance, and security.

Software faults occur through the following processes. A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, causing a failure. Not all defects will necessarily result in failures. For example, defects in dead code will never result in failures. A defect can turn into a failure when the environment is changed. Examples of these changes in environment include the software being run on a new computer hardware platform, alterations in source data, or interacting with different software. A single defect may result in a wide range of failure symptoms.

Input combinations and preconditions

A fundamental problem with software testing is that testing under *all* combinations of inputs and preconditions (initial state) is not feasible, even with a simple product. This means that the number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing. More significantly, non-functional dimensions of quality (how it is supposed to *be* versus what it is supposed to *do*)—usability, scalability, performance, compatibility, reliability—can be highly subjective; something that constitutes sufficient value to one person may be intolerable to another.

Software developers can't test everything, but they can use combinatorial test design to identify the minimum number of tests needed to get the coverage they want. Combinatorial test design enables users to get greater test coverage with fewer tests. Whether they are looking for speed or test depth, they can use combinatorial test design methods to build structured variation into their test cases. Note that "coverage", as used here, is referring to combinatorial coverage, not requirements coverage.

Related processes

Software verification and validation

Software testing is used in association with verification and validation:

- **Verification**: Have we built the software right? (i.e., does it implement the requirements).
- **Validation**: Have we built the right software? (i.e., do the deliverables satisfy the customer).

The terms verification and validation are commonly used interchangeably in the industry; it is also common to see these two terms incorrectly defined. According to the IEEE Standard Glossary of Software Engineering Terminology:

> Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.
>
> Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

According to the ISO 9000 standard:

- Verification is confirmation by examination and through provision of objective evidence that specified requirements have been fulfilled.
- Validation is confirmation by examination and through provision of objective evidence that the requirements for a specific intended use or application have been fulfilled.

Software quality assurance (SQA)

Software testing is a part of the software quality assurance (SQA) process. In SQA, software process specialists and auditors are concerned for the software development process rather than just the artifacts such as documentation, code and systems. They examine and change the software engineering process itself to reduce the number of faults that end up in the delivered software: the so-called "defect rate". What constitutes an "acceptable defect rate" depends on the nature of the software; A flight simulator video game would have much higher defect tolerance than software for an actual airplane. Although there are close links with SQA, testing departments often exist independently, and there may be no SQA function in some companies.

Software testing is a task intended to detect defects in software by contrasting a computer program's expected results with its actual results for a given set of inputs. By contrast, QA (quality assurance) is the implementation of policies and procedures intended to prevent defects from occurring in the first place.

White Box Testing

4.2 Internal and External Views of Testing

There are two views of the testing. The internal vies and external view. The internal view is also known as white box testing and the external view is also known as black box testing.

1. **Black box testing**
   The black box testing is used to demonstrate that the software functions are operational. As the name suggests in black box testing it is tested whether the input is accepted properly and output is correctly produced.

   The major focus of black box testing is on functions, operations, external interfaces, external data and information.

2. **White box testing**
   In white box testing the procedural details are closely examined. In this testing the internals of software are tested to make sure that they operate according to specifications and designs. Thus major focus of white box testing is on internal structures, logic paths, control flows, data flows, internal data structures, conditions, loops, etc.

4.3 White Box Testing

- The structural testing is sometime called as white box testing.
- In structural testing derivation of test cases is according to program structure. Hence knowledge of the program is used to identify additional test cases.
- Objective of structural testing is to exercise all program statements.

4.3.1 Condition Testing

To test the logical conditions in the program module the condition testing is used. This condition can be a Boolean condition or a relational expression.
The condition is incorrect in following situations.

1) Boolean operator is incorrect, missing or extra.
2) Boolean variable is incorrect.
3) Boolean parenthesis may be missing, incorrect or extra.
4) Error in relational operator.
5) Error in arithmetic expression.

The condition testing focuses on each testing condition in the program.

The branch testing is a condition testing strategy in which for a compound condition each and every true or false branches are tested.

The domain testing is a testing strategy in which relational expression can be tested using three or four tests.

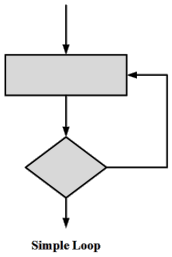
4.3.2 Loop Testing

Loop testing is a white box testing technique which is used to test the loop constructs. Basically there are four types of loops.

1) Simple loops :

The tests can be performed for n number of classes, where

1) n = 0 that means skip the loop completely.
2) n = 1 that means one pass through the loop is tested.
3) n = 2 that means two passes through the loop is tested.
4) n = m that means testing is done when there are m passes where men.
5) Perform the testing when number of passes are n - 1, n, n + 1.


Simple Loop

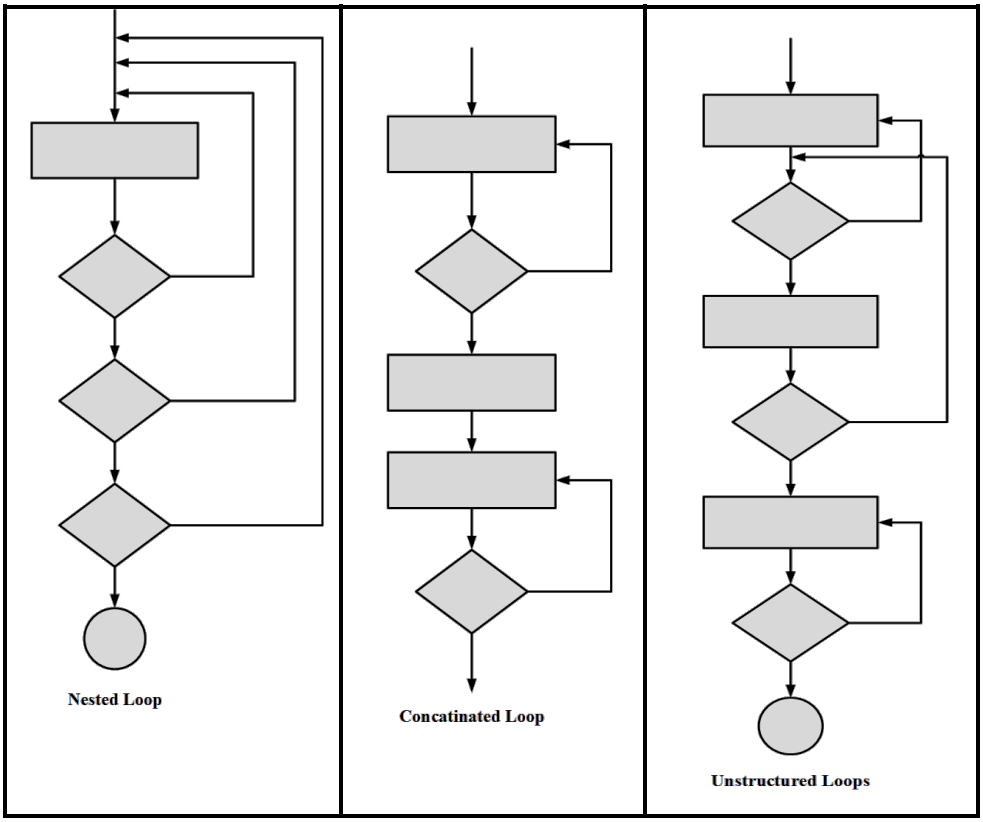
2) Nested loops

The nested loop can be tested as follows.

1) Testing begins from the innermost loop first. At the same time set all the other loops to minimum values.
2) The simple loop test for innermost loop is done.
3) Conduct the loop testing for the next loop by keeping the outer loops at minimum values and other nested loops at some specified value.
4) This testing process is continued until all the loops have been tested.

3) Concatenated loops

The concatenated loops can be tested in the same manner as simple loop tests.



4) Unstructured loops
The testing cannot be effectively conducted for unstructured loops. Hence these types of loops needs to be redesigned.

Advantages and Disadvantages of White box Testing

Advantages:
1) Each procedure can be tested thoroughly. The internal structures, data flows, logical paths, conditions and loops can be tested in detail.
2) It helps in optimizing the code.
3) White box testing can be easily automated.
4) Due to knowledge of internal coding structure it is easy to find out which type of input data can help in testing the application efficiently.

Disadvantages:
1) The knowledge of internal structure and coding is desired for the tested. Thus the skilled tester is required for whitebox testing. Due to this the testing cost is increased.
2) Sometimes it is difficult to test each and every path of the software and hence many paths may go untested.
3) Maintaining the white box testing is very difficult because it may use specialized tools like code analyzer, and debugging tools are required.
4) The missing functionality cannot be identified.

Basis Path Testing

4.3.3 Basis Path Testing

Path testing is a structural testing strategy. This method is intended to exercise every independent execution path of a program atleast once.

Following are the steps that are carried out while performing path testing.
[1] Design the flow graph for the program or a component.
[2] Calculate the cyclomatic complexity.
[3] Select a basis set of path.
[4] Generate test cases for these paths.

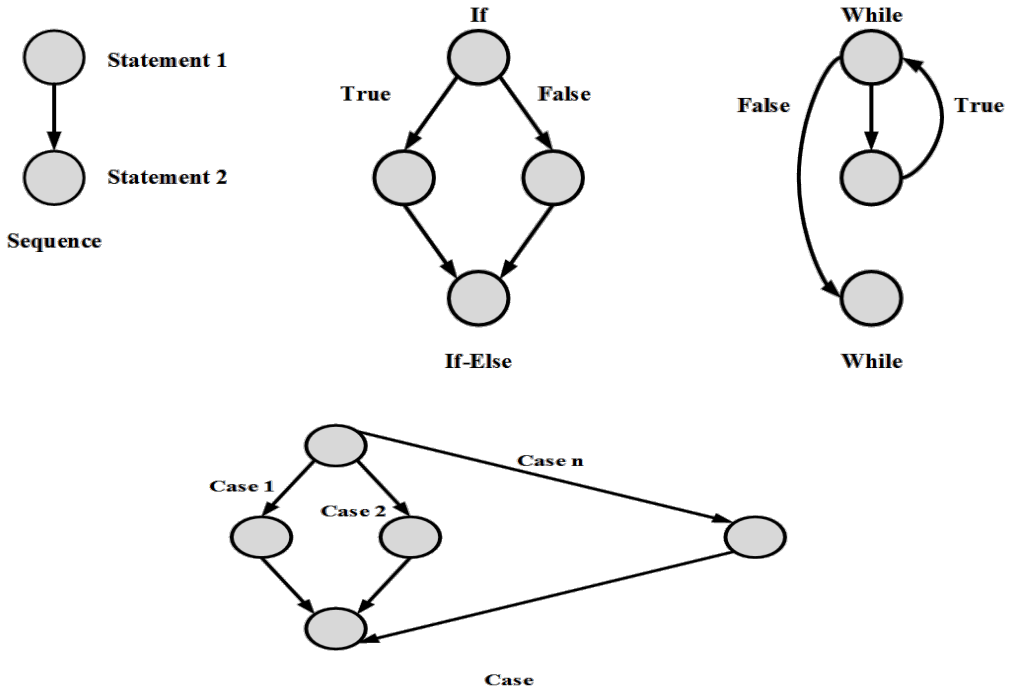
Step 1: Design the flow graph for the program or a component.
Flow graph is a graphical representation of logical control flow of the program. Such a graph consists of circle called a flow graph node which basically represents one or more procedural statements and arrow called as edges or links which basically represent control flow. In this flow graph the areas bounded by nodes and edges are called regions. Various notations used in flow graph are

Step 2: Calculate the cyclomatic complexity.
The cyclomatic complexity can be computed by three ways.

Cyclomatic complexity = Total number of regions in the flow graph = 4 (note that*! in above flow graph regions are given by shaded roman letters).

Cyclomatic complexity = E - N + 2 = 13 edges - 11 nodes + 2 =2+2=4

Cyclomatic complexity = P + 1= 3 + 1 = 4. There are 3 predicate (decision making) nodes : Nodes 3, 5 and 8.



Step 3: Select a basis set of path

The basis paths are

Path 1: 1, 2, 3, 4, 5, 6, 7, 11

Path 2: 1, 2, 3, 11

Path 3: 1, 2, 3, 4, 5, 8, 9, 3...

Path 4: 1, 2, 3, 4, 5, 8, 10, 3...

Step 4: Generate test cases for these paths.

After computing cyclomatic complexity and finding independent basis paths, the test cases has to be executed for these paths. The format for test case is –

Test case id	Test case name	Test case description	Test steps			Test case status (Pass/Fail)	Test Priority	Defect Severity
			Step	Expected	Actual			
1								
2								
: :								
n								

The test case for binary search can be written as -

Precondition
There should be list of elements arranged in ascending order. The element to be searched from the list, its value should be entered and will be stored in variable 'key'.

Black Box Testing

4.4 Black Box Testing

The black box testing is also called as behavioral testing.

- Black box testing methods focus on the functional requirements of the software. Test sets are derived that fully exercise all functional requirements.
- The black box testing is not an alternative to white box testing and it uncovers different class of errors than white box testing.

Why to perform black box testing?
Black box testing uncovers following types of errors.

1) Incorrect or missing functions
2) Interface errors
3) Errors in data structures
4) Performance errors
5) Initialization or termination errors

4.4.1 Equivalence Partitioning

It is a black box technique that divides the input domain into classes of data. From this data test cases can be derived.

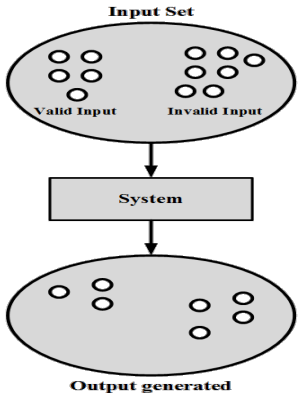
An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed.

In equivalence partitioning the equivalence classes are evaluated for given input condition. Equivalence class represents a set of valid or invalid states for input conditions.

Equivalence class guidelines can be as given below:

- If input condition specifies a range, one valid and two invalid equivalence classes are defined.
- If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
- If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined.
- If an input condition is Boolean, one valid and one invalid equivalence class is defined.

For example:
Area code: Input condition, Boolean - The area code may or may not be present. Input condition, range - Value defined between 200 and 700.
Password: Input condition, Boolean - A password may or may not be present.
Input condition, value - Seven character string.
Command: Input condition, set - Containing commands noted before.



4.4.2 Boundary Value Analysis (BVA)

Boundary value analysis is done to check boundary conditions.

A boundary value analysis is a testing technique in which the elements at the edge of the domain are selected and tested.

Using boundary value analysis, instead of focusing on input conditions only, the test cases from output domain are also derived.

Boundary value analysis is a test case design technique that complements equivalence partitioning technique.

Guidelines for boundary value analysis technique are

1) If the input condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.
2) If input condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.
3) If the output condition specified the range bounded by values x and y, then test cases should be designed with values x and y. Also test cases should be with the values above and below x and y.
4) If output condition specifies the number of values then the test cases should be designed with minimum and maximum values as well as with the values that are just above and below the maximum and minimum should be tested.
5) If the internal program data structures specify such boundaries then the test cases must be designed such that the values at the boundaries of data structure can be tested.

For example:

Integer D with input condition [- 2, 10],

Test values: - 2, 10, 11, - 1, 0

If input condition specifies a number values, test cases should developed to exercise the minimum and maximum numbers. Values just above and below this min and max j should be tested.

Enumerate data E with input condition: {2, 7, 100, 102}

Test values: 2, 102, - 1, 200, 7

Comparison between Black Box Testing and White Box Testing

Black box testing	White box testing
Black box testing is called behavioral testing.	White box testing is called glass box testing.
Black box testing examines some fundamental aspect of the system with little regard for internal logical structure of the software.	In white box testing the procedural details, all the logical paths, all the internal data structures are closely examined.
During black box testing the program cannot be tested 100 percent.	White box testing lead to test the program thoroughly.
This type of testing is suitable for large projects.	This type of testing is suitable for small projects.

Advantages and Disadvantages of Black box Testing

Advantages:

1) The black box testing focuses on fundamental aspect of system without being concerned for internal logical structure of the software.
2) The advantage of conducting black box testing is to uncover following types of errors.
   - Incorrect or missing functions
   - Interface errors
   - Errors in external data structures
   - Performance errors
   - Initialization or termination errors

Disadvantages:

1) All the independent paths within a module cannot be tested.
2) Logical decisions along with their true and false sides cannot be tested.
3) All the loops and the boundaries of these loops cannot be exercised with black box testing.
4) Internal data structure cannot be validated.

Comparison between Black Box Testing vs White Box Testing:

BLACK BOX TESTING	WHITE BOX TESTING
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure r the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software.	It is structural test of the software.
This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: search something on google by using keywords	**Example:** by input to check and verify loops

Some of the **advantages of black-box testing** are:

1. Efficient for large segments of code
2. Code access is not required
3. Separation between user's and developer's perspectives

Some of the **disadvantages of black-box testing** are:

1. Limited coverage since only a fraction of test scenarios is performed
2. Inefficient testing due to tester's luck of knowledge about software internals
3. Blind coverage since tester has limited knowledge about the application

Some of the **advantages of white-box testing** are:

1. Efficient in finding errors and problems
2. Required knowledge of internals of the software under test is beneficial for thorough testing
3. Allows finding hidden errors
4. Programmers introspection
5. Helps optimizing the code
6. Due to required internal knowledge of the software, maximum coverage is obtained

Some of the **disadvantages of white-box testing** are:

1. Might not find unimplemented or missing features
2. Requires high level knowledge of internals of the software under test
3. Requires code access

Testing Strategy

4.5 Testing Strategy

We begin by “testing-in-the-small” and move toward “testing-in-the-large”
Various testing strategies for conventional software are

- Unit testing
- Integration testing
- Validation testing'
- System testing

Unit testing

In this type of testing techniques are applied to detect the errors from each software component individually.

Integration testing

It focuses on issues associated with verification and program construction as components begin interacting with one another

Validation testing

It provides assurance that the software validation criteria (established during requirements analysis) meets all functional, behavioral, and performance requirements.

System testing

In system testing all system elements forming the system is tested as a whole.

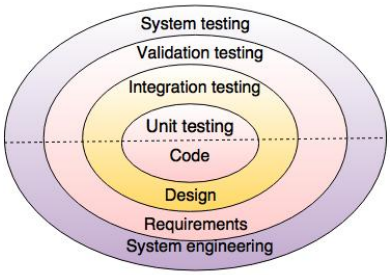


Fig. - Testing Strategy

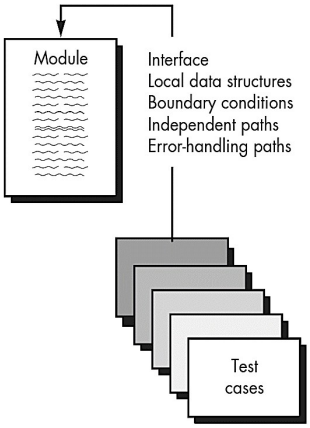
4.6 Unit Testing

In unit testing the individual components are tested independently to ensure their quality.

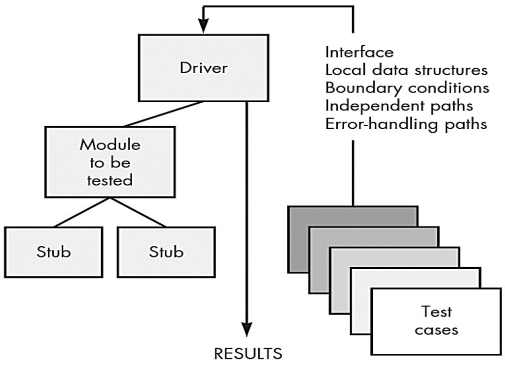
The focus is to uncover the errors in design and implementation.

The various tests that are conducted during the unit test are described as below.

1. Module interfaces are tested for proper information flow in and out of the program.
2. Local data are examined to ensure that integrity is maintained.
3. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
4. All the basis (independent) paths are tested for ensuring that all statements in the module have been executed only once.
5. All error handling paths should be tested.



6. Drivers and stub software need to be developed to test incomplete software. The "driver" is a program that accepts the test data and prints the relevant results. And the "stub" is a subprogram that uses the module interfaces and performs the minimal data manipulation if required.



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

4.7 Integration Testing

A group of dependent components are tested together to ensure their quality of their integration unit.

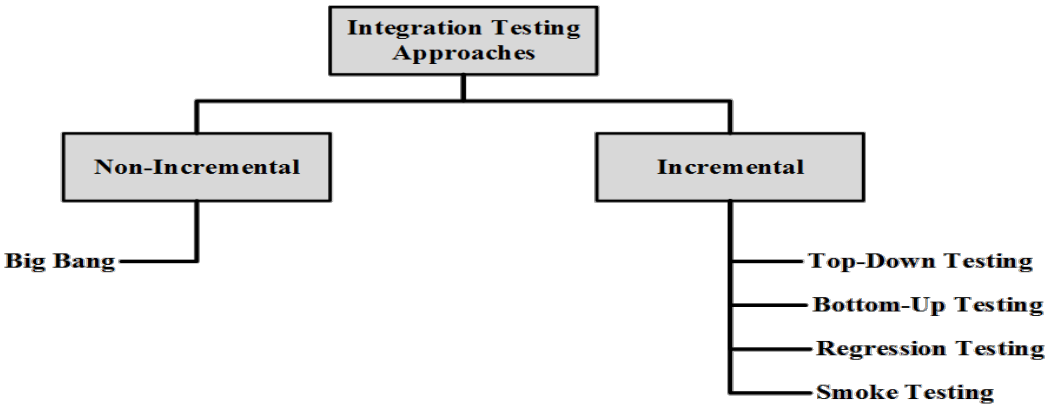
The objective is to take unit tested components and build a program structure that has been dictated by software design.

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.

Big-Bang

Advantage:

- This approach is simple.

Disadvantages:

- It is hard to debug.
- It is not easy to isolate errors while testing.
- In this approach it is not easy to validate test results.
- After performing testing, it is impossible to form an integrated

system. An incremental construction strategy includes

[1] Top down integration
[2] Bottom up integration
[3] Regression testing
[4] Smoke testing

4.7.1 Top-Down Integration Testing

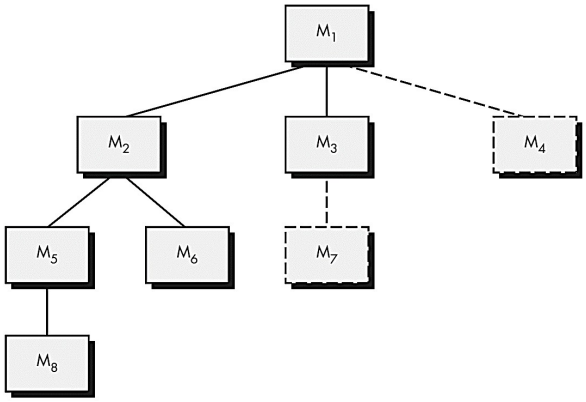
Top down testing is an incremental approach in which modules are integrated by moving down through the control structure.

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 Ml then we will integrate M2 then M3, M4, M5, M6 and then M7.

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

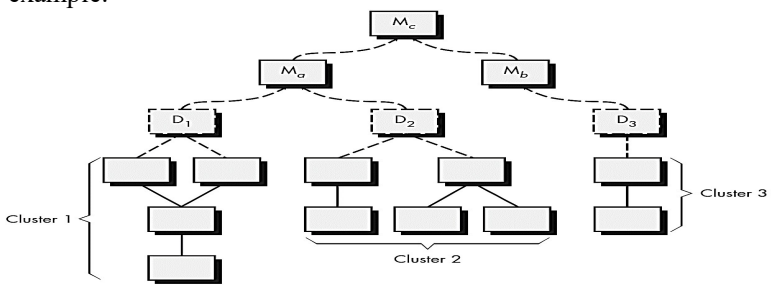
4.7.2 Bottom Up integration Testing

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 sub-function.
[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:



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.

4.7.3 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.

4.7.4 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

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

Validation Testing

4.8 Validation Testing

The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.

Validation Testing ensures that the product actually meets the client's needs. It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.

It answers to the question, Are we building the right product?

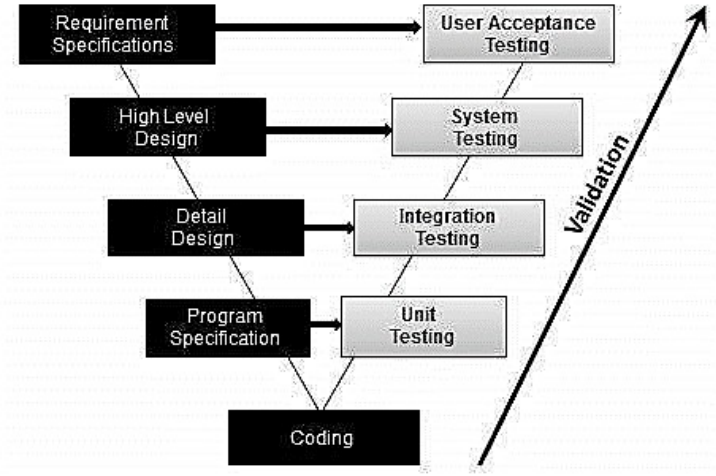
Validation Testing - Workflow

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of 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 behavior 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 configurations are developed as per requirements. This review is called configuration review or audit.

4.8.1 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 reports them to developer. The developer then makes appropriate modification.

System Testing and Debugging

4.9 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 behavior 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.

4.9.1 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 re-initialization, checkpoint mechanisms, data recovery and restart are verified.

4.9.2 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.

4.9.3 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.

4.9.4 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.

4.10 Debugging

Debugging is a process of removal of a defect. It occurs as a consequence of successful testing.

Debugging process starts with execution of test cases. The actual test results are compared with the expected results. The debugging process attempts to find the lack of correspondence between actual and expected results. The suspected causes are identified and additional tests or regression tests are performed to make the system to work as per requirement.

Common approaches in debugging are :

[1] Brute force method

The memory dumps and run-time traces are examined and program with write statements is loaded to obtain clues to error causes.

In this method "Let computer find the error" approach is used.

This is the least efficient method of debugging.

[2] Backtracking method

This method is applicable to small programs.

In this method, the source code is examined by looking backwards from symptom to potential causes of errors.

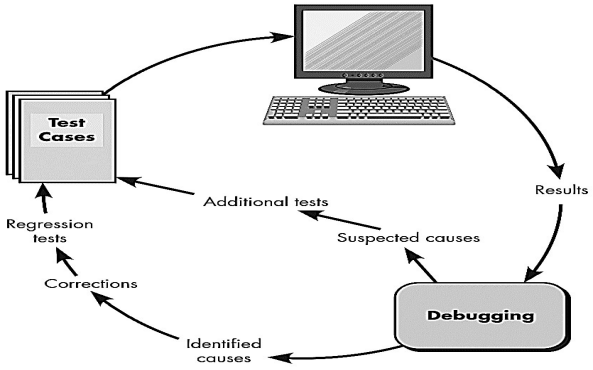
[3] Cause elimination method

This method uses binary partitioning to reduce the number of locations where errors can exist.

Difficulties in Debugging

Following are some reasons that reveal why debugging is so difficult -

1. The symptoms of bug may be present at some part (module) of the program and the effect might be seen in some other module of the program. Hence tracing out the location of symptom becomes difficult.
2. Symptoms may be caused by software developers during the development process. Such symptoms are difficult to trace out.
3. The symptom may appear due to timing problems instead of processing problem
4. If the developer corrects some error then the symptom may disappear temporarily.
5. The symptom can appear if some in-accuracies in the program are simply rounded off.
6. In real time systems, it is not possible to accurately reproduce the input conditions and this may lead to symptoms of bugs.
7. The symptom may appear periodically. Such things normally occur embedded systems in which hardware and software is coupled.
8. In distributed systems number of tasks are running on several distinct processors which may lead to symptoms.



Techniques in debugging

- Print debugging
- Remote debugging
- Post-mortem debugging
- "Wolf fence" algorithm.
- Delta Debugging
- Saff Squeeze

Testing Vs. Debugging

Testing	Debugging
Testing is a process in which the bug is identified	Debugging is the process in which the bug or error is corrected by the programmer
In testing process, it is identified where the bug Occurs.	In debugging The root cause of error is Identified.
Testing starts with the execution results from the test cases.	Debugging starts after the testing process.

Software Implementation Techniques

4.11 Software Implementation Techniques

After detailed system design we get a system design which can be transformed into implementation model. The goal coding is to implement the design in the best possibly manner. Coding affects both testing and maintenance very deeply. The coding should be | done in such a manner that the instead of getting the job of programmer simplified the task of testing and maintenance phase should get simplified.

Various objectives of coding are -

[1] Programs developed in coding should be readable.
[2] They should execute efficiently.
[3] The program should utilize less amount of memory.
[4] The programs should not be lengthy.

If the objectives are clearly specified before the programmers then while coding they try to achieve the specified objectives. To achieve these objectives some program principles must be followed.

4.11.1 Coding Practices

There are some commonly used programming practices that help in avoiding common errors. These are enlisted below -

[1] Control construct

The single entry and single exit constructs need to be used. The standard control constructs must be used instead of using wide variety of controls.

[2] Use of gotos

The goto statements make the program unstructured and it also imposes overhead on compilation process. Hence avoid use of goto statements as far as possible and another alternative must be thought of.

[3] Information hiding

Information hiding should be supported as far as possible. In that case only access functions to the data structures must be made visible and the information present in it must be hidden.

[4] Nesting

Nesting means defining one structure inside another. If the nesting is too deep then it becomes hard to understand the code. Hence as far as possible - avoid deep nesting of the code. Note that after removing of nesting the code becomes more readable but still the same output can be achieved.

[5] User defined data types

Modern programming languages allow the user to use defined data types as the enumerated types. Use of user defined data types enhances the readability of the code.

For example: In C we can define the name of the days using enumerated data type

[6] Module size

There is no standard rule about the size of the module but the large size of the module will not be functionally cohesive.

[7] Module interface

Complex module interface must be carefully examined. A simple rule of thumb is that the module interface with more than five parameters must be broken into multiple modules with simple interface.

[8] Side effects

Avoid obscure side effects. If some part of the code is changed randomly then it will cause some side effect. For example if number of parameters passed to the function is changed then it will be difficult to understand the purpose of that function.

[9] Robustness

The program is said to robust if it does something even though some unexceptional condition occurs. In such situations the programs do not crash but it exits gracefully.

[10] Switch case with defaults

The choice being passed to the switch case statement may have some unpredictable value, and then the default case will help to execute the switch case statement without any problem. Hence it is a good practice to always have default case in the switch statement.

[11] Empty catch block

If the exception is caught but if there is no action then it is not a good practice. Therefore take some default action even if it is just writing some print statement, whenever the exception is caught.

[12] Empty if and while statements

In the if and while statements some conditions are checked. Hence if we write some empty block on checking these conditions then those checks are proved to be useless checks. Such useless checks should be avoided.

[13] Check for read return

Many times the return values that we obtain from the read functions are not checked because we blindly believe that the desired result is present in the corresponding variable when the read function is performed. But this may cause some serious errors. Hence the return value must be checked for the read operation.

[14] Return from Finally Block

The return value must come from finally block whenever it is possible. This helps in distinguishing the data values that are returning from the try-catch statements.

[15] Trusted Data sources

Counter check should be made before accessing the input data. For example while reading the data from the file one must check whether the data accessed is NULL or not.

[16] Correlated Parameters

Many often there occurs co-relation between the data items. It is a good practice to check these co-relations before performing any operation on those data items.

[17] Exceptions Handling

If due to some input condition if the program does not follow the main path and follows an exceptional path. In such a situation, an exceptional path may get followed. In order to make the software more reliable, it necessary to write the code for execution.

4.11.2 Coding Standards

Any good software development approach suggests to adhere to some well-defined standards or rules for coding. These rules are called coding standards.

[1] Naming Conventions

Following are some commonly used naming conventions in the coding

- Package name and variable names should be in lower case.
- Variable names must not begin with numbers.
- The type name should be noun and it should start with capital letter.
- Constants must be in upper case (For example PI, SIZE)
- Method name must be given in lower case.
- The variables with large scope must have long name. For example count_total, sum, Variables with short scope must have short name. For example i,j.
- The prefix is must be used for Boolean type of variables. For example isEmpty or isFull

[2] Files

Reader must get an idea about the purpose of the file by its name. In some programming language like Java -

- The file extension must be java.
- The name of the file and the class defined in the file must have the same na
- Line length in the file must be limited to 80 characters.

[3] Commenting/Layout

Comments are non-executable part of the code. But it is very important because it enhances the readability of the code. The purpose of the code is to explain the logic of the program.

- Single line comments must be given by //
- For the names of the variables comments must be given.
- A black of comment must be enclosed within /* and */.

[4] Statements

There are some guidelines about the declaration and executable statements.

- Declare some related variables on same line and unrelated variables on another line.
- Class variable should never be declared public.
- Make use of only loop control within the for loop.
- Avoid make use of break and continue statements in the loop.
- Avoid complex conditional expressions. Make use of temporary variables instead.
- Avoid the use of do...while statement.

Advantages of Coding Standards

- Coding standard brings uniform appearance in system implementation.
- The code becomes readable and hence can be understood easily.
- The coding standard helps in adopting good programming practices.

4.11.3 Refactoring

A change in the code is very common in the software development process. Due to change in the requirements or due to addition of new functionality the changes occur in the coding.

For accommodating those changes in the code we need to change the design. If we plan to code as per the changed design, then code becomes very complex. Productivity and quality deteriorates.

Refactoring is the technique used to improve the code and avoid the design decay with time.

Refactoring is done during the coding. It also plays an important role in test driven development in code improvement step.

Refactoring is defined as a change made to the internal structure of software for better understanding and performing cheaper to modifications without changing system's behavior.

The basic objective of refactoring is to improve the design of the system. Refactoring is basically done in order to improve the design of code that already exists.

As result of refactoring one of the following things may occur

- The coupling may get reduced.
- Cohesion may get increased
- The open-closed principle may get followed strictly.

Refactoring involves the changes in the code. If there is a requirement for code change then the refactoring occurs.

The main risk of refactoring is that existing working code may break due to the changes being made (may lose the objective of the code). This is the main reason why most often refactoring is not done.

In order to mitigate or avoid this risk following two rules must be followed.

Rule 1: Re-factor in small steps

Rule 2: For testing the existing functionalities make use of test scripts

By following these two rules the bugs in the refactoring can be easily identified and corrected.

In each refactoring only small change is made but the series of refactoring makes a significant transformations in the program structure.

With refactoring design as well as coding gets improved. Because with refactoring the quality of design improves which helps in making the useful changes in the coding.

Due to refactoring, the cost of testing and debugging gets reduced. The efforts required in making changes in the code get reduced and we get an improved quality code.

Thus the purpose of refactoring is to create a long lasting and healthy code.

4.12 Maintenance and Reengineering

- Software maintenance is an activity in which program is modified after it has been put into use.
- In software maintenance usually it is not preferred to apply major software changes to system's architecture.
- Maintenance is a process in which changes are implemented by either modifying the existing system's architecture or by adding new components to the system.

Need for Maintenance

The software maintenance is essential because of following reasons:

1. Usually the system **requirements are changing** and to meet these requirements some changes are incorporated in the system.
2. There is a strong relationship between system and its environment. When a system is installed in an environment, it changes that environment. This ultimately changes the system requirements.
3. The maintained system remains useful in their working environment.

- Maintenance is applicable to software developed using any software life cycle model. The system changes and hence maintenance must be performed in order to :

a. Correct faults.
b. Improve the design.
c. Implement enhancement.
d. Interface with other systems.
e. Adoption of environment (different hardware, software, system features etc.).
f. Migrate legacy software.
g. Replacement of old software by new software, o

- In software maintenance report four key characteristics should be mentioned.
  a. Maintaining control over the software's day to day functions.
  b. Maintaining control over software modification.
  c. Reparing of functions.
  d. Performance degradation should be avoided.

Types of Software Maintenance

1. **Corrective maintenance** - Means the maintenance for correcting the software faults.
2. **Adaptive maintenance** - Means maintenance for adapting the change in environment (different computers or different operating systems).
3. **Perfective maintenance** - Means modifying or enhancing the system to meet the new requirements.
4. **Preventive maintenance** - Means changes made to improve future maintainability.

- According to Lientz and Swanson (in 1980) and Nosek and Palvia (in 1990), if new requirements are added then it needs lot of efforts to maintain such systems.
- Above Fig. 4.12.1 shows that making some modifications in the existing system or adding some new functionalities is very costly from maintenance point of view.
- Nearly 65 % of efforts are required for such maintenance.
- If the operating environment gets changed then 18 % of maintenance cost will be required. But repairing or correcting faults is less costly which may cost around 17 % of total effort cost.

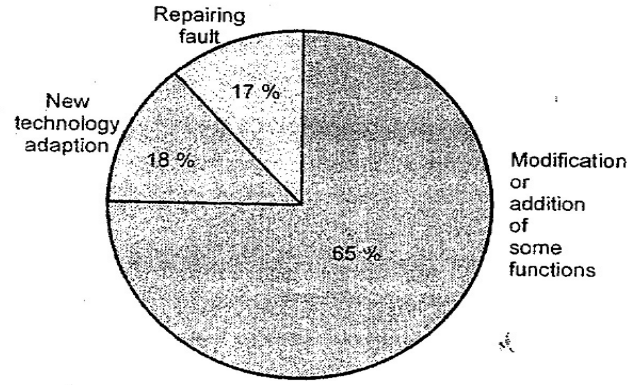


Fig 4.12.1 Maintenance cost

Software Maintenance Process

The software evolution process is dependent upon the type of software being maintained
The software maintenance process can be as shown below.

1. In the maintenance process initially the request for change is made.
2. **Change management** - In this phase the status of all the change **requests is** identified, described.
3. **Impact analysis** - Following activities are performed in this phase
   a. Identify all systems and system products affected by a change request.
   b. Make an estimate of the resources needed to effect the change,
   c. Analyze the benefits of the change.
4. **System release planning** - In this phase the schedule and contents **of software** release is planned. The changes can be made to all types of software maintenance.

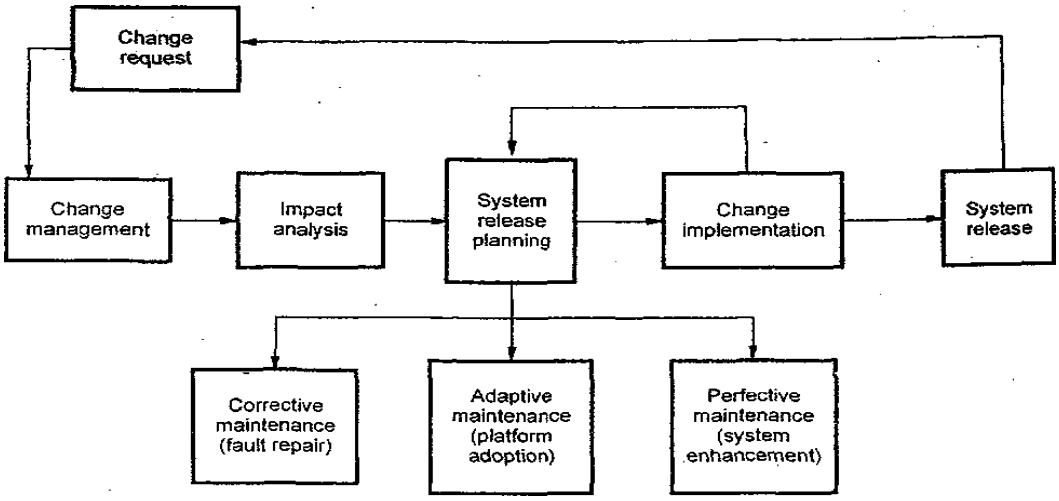


Fig. 4.13.2 Maintenance process

5. **Change implementation** - The implementation of changes can be done by first designing the changes, then coding for these changes and finally testing the changes. Preferably the regression testing must be performed while testing the changes.
6. **System release** - During the software release i) Documentation ii) Software iii) Training iv) Hardware changes v) Data conversion should be described.

Factors affecting maintenance costs

- Module Independence - This is the ability to modify one part of the system.
- Programming Language - For the higher level of the language, the maintenance is cheaper.
- Programming Style - The way in which a program is written makes difference in the cost.
- Program Validation and Testing - The more time and effort spent on design validation and program testing, the fewer errors and the less the need for corrective maintenance.
- Quality of Program Documentation - The better the documentation, the easier it is to maintain.
- Configuration Management Techniques - Keeping track of all the system documents and ensuring they are consistent is a major cost of maintenance
- Application Domain - If the application domain is not well understood then the chances of errors are more. This causes high cost on maintenance.
- Staff Stability - Maintenance costs are reduced if developers have to maintain their own systems.
- Age of the System - Older systems are difficult to maintain.
- Dependence of the System on the External Environment - If the system is highly dependant upon external environment then lot much of maintenance is needed.
- Hardware Stability - If the hardware platform will not change over the life of the system then the maintenance will not be needed.

Issues in Software Maintenance

1. **Technical** - This is a key issue in software maintenance. Technical maintenance is based on following factors such as limited understanding of system, testing, impact analysis, maintainability.
2. **Management** - Management issue includes organizational issues, staffing probelm, process issue, organizational structure, outsourcing.
3. **Cost estimation**—This is one of the major issues in software maintenance. It is based on cost, experience of projects.
4. **Software maintenance measurement** - The software measurement factors such as size effort, schedule, quality, understandability, resource utilization, design complexity, reliability and fault type distribution.

4.13 Business Process Reengineering

Definition: The Business Process re-engineering (BPR) is the implementation of radical change in the business process to achieve breakthrough results.
The business process is a set of tasks that are performed in order to achieve the desired business outcome.
For example-Purchasing services, designing a new product are the business processes that except some business outcome.

4.13.1 BPR Model

- The BPR model is evolutionary process model. It is iterative in nature.
- There are six activities carried out in this model. Refer Fig. below

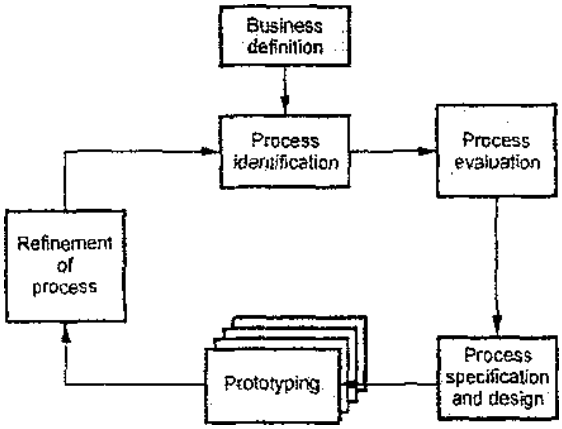


Fig. BPR Model

1. **Business Process Definition :**
   - The processes are defined based on business goals.
   - The business goals are defined using the key factors
     a. Cost reduction
     b. Quality improvement
     c. Time reduction and
     d. Personnel development.
2. **Process Identification :**
   - The critical processes are identified.
   - They are prioritized according to their need for change.
3. **Process Evaluation :**
   - The existing processes are analyzed.
   - Various tasks that constitute a process are identified.
   - The time and cost required by these tasks are measured.
   - The quality performance issues are identified and isolated
4. **Process Specification and Design:**
   - The use cases are prepared for each process that need to be redesigned in BPR.
   - Each use case captures the scenario that give out some outcome to the customer.
   - On analysing the use cases, new task are redesigned if required.
5. **Prototyping**
   - Before integrating it to the system, the redesigned process is prototyped for testing purpose.
6. **Refinement and instantiation**
   - The feedback for each prototype in BPR model is collected.
   - The processes are refined based on the feedback.

4.14 Reengineering Process Model

Software reengineering is a process of modifying the system for main' purpose. The software reengineering process model depicts six activities as shown in Fig

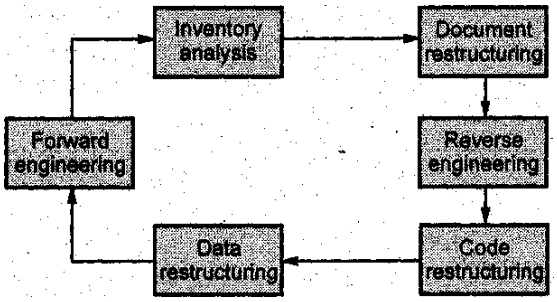


Fig. 4.14.1 Software reengineering process model

Inventory Analysis:

- The software organization possesses the inventory of all the required applications.
- This inventory should be revisited periodically.
- The resources can be allotted to the candidate applications for reengineering work.
- According to the changes in the candidate applications their status need to updated.

Document Restructuring:

- Documents are essential part of software project lifecycle. But inadequate inappropriate approach of documentation leads to document restructuring activity.
- There are three alternatives that can be chosen for document restructuring –
- Instead of having time consuming documentation, remain with weak documentation.
- Update poor documentation if needed.
- For the critical systems, rewrite the documents if needed.

Reverse Engineering:

- Reverse engineering is the process of design recovery.
- In reverse engineering the data, architectural and procedural information is extracted from a source code.
- The reverse engineering creates a representation of the program at a higher level of abstraction than the source code.

Code Restructuring:

- This is an activity in which the code is analyzed.
- If the programming constructs are violated then that code is restructured.
- Sometimes the code is rewritten in modern programming language.
- The resultant restructured code is tested and reviewed.

Data Restructuring:

- Data restructuring is large scale reengineering activity.
- In this activity, data architecture is dissected and necessary data models are refined.
- If data structures are weak then data are reengineered.
- The changes in data demand for changes in architecture or code.

Forward Engineering

- Forward engineering is a process in which the design information is recovered from the existing software and the overall quality of the code is improved.
- Many times new functionalities are added in the existing system to improve overall performance.

4.15 Reverse and Forward Engineering

Reverse engineering is the process of design recovery. In reverse engineering the data, architectural and procedural information is extracted from a source code.

There are three important issues in reverse engineering.

1. **Abstraction level**: This level helps in obtaining the design information from the source code. It is expected that abstraction level should be high in reverse engineering. High abstraction level helps the software engineer to understand the program.
2. **Completeness level**: The completeness means detailing of abstract level. The completeness decreases as abstraction level increases.
   For example - From a given source code listing one can easily develop a complete procedural design representation. But it is very difficult to develop complete set of data flow diagrams or entity relationship diagram. The completeness in reverse engineering develops the interactivity. The term interactivity means the degree to which the human is integrated with automated tools to create effective reverse engineering process. As the abstraction level increases the interactivity must increase to bring the completeness.
3. **Directionality level**: Directionality means extracting the information from source code and give it to software engineer. The directionality can be one way or two way. The one way directionality means extracting all the information from source code and give it to software engineer. The two way directionality means the information taken from source code is fed to a re-engineering tool that attempts to restructure or regenerate old programs.

Reverse Engineering Process

- Initially the dirty source code or unstructured source code is taken and processed and the code is restructured. After restructuring process the source code becomes clean.
- The core to reverse engineering is an activity called extract abstractions.
- In extract abstraction activity, the engineer must evaluate older programs and extract information about procedures, interfaces, data structures or databases used.
- The output of reverse engineering process is a clear, unambiguous final specification obtained from unstructured source code. This final specification helps in easy understanding of source code.

Difference between Software Engineering and Reverse Engineering

Sl.No	Software engineering	Reverse engineering
1	Software engineering is a, discipline in which theories, methods and tools are applied to develop a professional software product.	Reverse engineering is a process in which the dirty or unstructured code is-taken, processed and it is restructured.
2	Initially only user requirements are available for software engineering process.	A dirty or unstructured code is available initially
3	This process starts by understanding user requirements	This process starts by understanding the j existing unstructured code.
4	The software. engineering is conducted using, requirement gathering; analysis, design, implementation and testing.	The reverse engineering is conducted using restructuring the code, cleaning it, ! by extracting the abstractions. After refinement and simplification of the code final code gets ready.
5	It is simple and straightforward approach.	It is complex because cleaning the dirty .: or unstructured code requires more ; efforts.

6	Documentation or specification of the product is useful to the end-user	Documentation or specification of the product is useful to the developer

Difference between Reverse Engineering and Re-engineering

Sl.No	**Reverse engineering**	**Re-engineering**
1	Reverse engineering is a process of , finding out how a product works from already created software system	Re-engineering is to observe the software system and build it again for better use.
2	In reverse engineering, the source code is re-created from the compiled code.	In re-engineering, new piece of code with similar or better functionality than the existing one - is created,
3	Reverse engineering w carried out for trying to understand inner working of the artifact with availability of any documents	Re-engineering is carried out for designing something again. Many times from scratch.

Forward Engineering

If the poorly designed and implemented code is to be modified then following alternatives can be adopted -

1. Make lot of modifications to implement the necessary changes.
2. Understand inner workings of the program in order to make the necessary modifications.
3. Redesign, recode and test ***small*** modules of software that require modifications.
4. Completely redesign, recode and test the entire program using re-engineering tool.

Definition : Forward engineering is a process that makes use of software engineering principles, concepts and methods to re-create an existing application. This re-developed program extends the capabilities of old programs.

Forward Engineering for Object Oriented Architectures

- Forward engineering is a process of re-engineering conventional software into the object oriented implementation.
- Following are the steps that can be applied **J**OT forward engineering the conventional software -
  1. Existing software is reverse engineered in order to create data, functional, and behavioural models.
  2. If existing system extends the functionality of original application then **use cases** can be created.
  3. The data models created in this process are used to create the base for classes.
  4. Class Hierarchies, object-relationship models, object behavioral models, and subsystems are defined
- During this forward engineering process, algorithms and data structures are reused from existing conventional application.

Difference between Forward and Reverse Engineering

- Forward engineering is a process of constructing a system for specific purpose.
- Reverse engineering is a process of de-constructing a system in order to extend the functionalities or in order to understand the working of the system.

Part – A

Software testing fundamentals

1. **What are static and dynamic software testing?**
   The verification activities fall into the category of static testing. During static testing, you have a checklist to check whether the work you are doing is going as per the set standards of the organization.

   The dynamic testing is a method in which the actual testing is done to uncover all possible errors. Various testing strategies such as unit testing, integration testing, validation testing, system testing can be applied while performing dynamic testing.

2. **Why testing is important with respect to software?**
   The purpose of software testing is to ensure whether the software functions appear to be working according to specifications and performance requirements. This ultimately helps in improving the overall quality of the software.

3. **Write down generic characteristics of software testing.**
   Following are the characteristics of software testing :
   - Testing should be conducted by the developer as well as by some independent group.
   - The testing must begin at the component level and must work outwards towards the integration testing.
   - Different testing must be appropriate at appropriate times.

4. **Write the best practices for "Coding". [MJ/ND 2015]**
   The following are the practices to be followed during the coding phase in order to produce high quality software
   - Clear definition of purpose.
   - Simplicity of understand.
   - Ruggedness (difficult to misuse or damage).
   - Early availability (delivered on time when needed).
   - Reliability.
   - Extensibility.
   - Minimum cost to develop.
   - Conformity to any relevant standards.
   - Clear

White Box Testing

5. **Which is called as a glass box testing? What is the objective of this?**
   Glass box testing is a testing technique that examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

6. **What is white box testing and what is the difficulty while exercising it?**
   In white box testing the procedural details are closely examined. In this testing the internals of software are tested to make sure that they operate according to specifications and designs. Thus major focus of white box testing is on internal structures, logic paths, control flows, data flow, internal data structures, conditions, loops, etc.

   This kind of testing will be exhaustive testing. This type of testing will keep the processor continuously busy. And software developer will find it difficult to manage the schedule of the project with white box testing. The white box testing is impossible for large system.

7. **What are the various Techniques used in White Box Testing?**
   - **Statement Coverage -** This technique is aimed at exercising all programming statements with minimal tests.
   - **Branch Coverage -** This technique is running a series of tests to ensure that all branches are tested at least once.
   - **Path Coverage -** This technique corresponds to testing all possible paths which means that each statement and branch is covered.

Basis Path Testing

8. **Define Basis Path Testing**

   In software engineering, **basis path testing**, or structured **testing**, is a white box method for designing test cases. The method analyzes the control flow graph of a program to find a set of linearly independent **paths** of execution

9. **What are the classes of loops that can be tested?**

   Various classes of loop testing are -
   - Simple Loop
   - Nested Loop
   - Concatenated Loop
   - Unstructured Loop

10. **What is Flow Graph needed to perform Basis Path Testing?**

    In order to compute logical complexity measure, V(G), of a code, testers perform the basis path testing. The value of V(G), defines the maximum number of test cases to be designed by identifying basis set of execution paths to ensure that all statements are executed at least once.

11. **Define cyclomatic complexity.**

    Cyclomatic complexity is kind of software metric that gives the quantitative measure of logical complexity of a program. In basis path testing method the cyclomatic complexity defines the number of independent paths of the program structure.

12. **What is the purpose of calculating cyclomatic complexity?**

    The cyclomatic complexity defines the number of independent paths in the basis set of the program that provides the upper bound for the number of tests that must be conducted to ensure that all the statements have been executed at least once.

Black Box Testing

13. **Define equivalence partitioning.**

    In equivalence partitioning the equivalence classes are evaluated for given input condition. Equivalence class represents a set of valid or invalid states for input conditions.

    It is a black-box technique that divides the input domain into classes of data. From this data test cases can be derived.

14. **What is behavioral testing?**

    The black box testing is also called as behavioral testing.
    Black box testing methods focus on the functional requirements of the software. Tests sets are derived that fully exercise all functional requirements.

Testing Strategy

15. What is the objective of unit testing?

The objective of unit testing is to test individual components independently to ensure their quality. Thus the focus is to uncover the errors in design and implementation.

16. List out the data structure errors identified during the unit testing.

Various data structure errors that can be identified during the unit testing are -

[1] Incorrect arithmetic precedence.
[2] Mixed mode operations.
[3] Precision inaccuracy.
[4] Comparison of different data types.

17. In unit testing of a module, it is found for a set of test data, at maximum 90 % of the code aione were tested with the probability of success 09. What is the reliability of the module? The reliability of the module is at the most 0.81. For the given set of test data the 90 % of the code is tested and probability of success is given as 0.9 i.e. 90 %. Hence Reliability = (90/100)^0.9 = 0.81

18. What are the steps for top-down integration?

Following are the steps in top-down integration.

[1] The main control module is used as a test driver and stubs are substituted for all the 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 search 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.

19. What is partial integration testing?

Partial integration testing verifies correct data movement from one external system to another.
For instance: If there is one system that brings data from single external system and then sends the data to several other external systems. Then partial test of this system would be to test whether the data can be sent to one of the external systems.

Validation Testing

20. Distinguish between verification and validation. [ND 2017/2016, A/M 2018]

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

21. List out the advantages of carrying out Validation Testing

During verification if some defects are missed then during validation process it can be caught as failures.

If during verification some specification is misunderstood and development had happened then during validation process while executing that functionality the difference between the actual result and expected result can be understood.

Validation is done during testing like feature testing, integration testing, system testing, load testing, compatibility testing, stress testing, etc.

Validation helps in building the right product as per the customer's requirement and helps in satisfying their needs.

22. What are the areas Validation Testing addresses?

Validation testing provides answers to questions such as:

- Does the software fulfill its intended use?
- Is the company building the right product?
- Can the project be properly implemented?
- Are the documents in line with the development process?

System Testing and Debugging

23. List out the types of system tests.

Various types of system tests are

- Recovery testing
- Security testing
- Stress testing
- Performance testing

24. Write the steps involved in testing real time systems.

Following are the steps involved in testing real time systems.

Task testing: This is the first step in testing real time systems. In this step, each task is independently tested. Task testing uncovers the errors in logic and function. But it does not find errors from behavior.

Behavioral testing: With the help of automated tool the behavior of the real time system is simulated. Then it becomes possible to examine the behavior of the system as a series of external events.

Inter task testing: After finding and removing the errors from individual task and behavior of the system the next step is to obtain errors from asynchronous tasks that may communicate with one another. Such type of testing is called inter task testing.

System testing: When all the required hardware and software is integrated then a full range of testing is carried out in order to uncover errors at hardware and software interface. As real time system is based on interrupts therefore testing the Boolean events is necessary.

25. Distinguish between stress and load testing.

Load testing is conducted to measure system performance based on volume of users. On the other hand the stress testing is conducted for measuring the breakpoint of a system when extreme load is applied. In load testing the expected load is applied to measure the performance of the system whereas in stress testing the extreme load is applied.

26. What is a Big-Bang approach?

Big-Bang approach is used in non-incremental integration testing. In this approach of integration testing, all the components are combined in advance and then entire program is tested as a whole.

27. How regression and stress tests are performed?

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.

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.

28. What is the need for Regression Testing? [M/J 2015]

The following are the main reasons why regression testing is needed,

- Understand your business area
- Create, validate and maintain the artifacts
- Understand the test cases required to build high coverage for the business area
- Build the test suite in priority order

29. What is stress testing?

Stress testing is a testing for a system which 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.

30. Justify why 100 % testing is impossible.

It is not possible to test all inputs or all permutations and combinations of all inputs. It is not even possible to test all paths of even a moderate system. For example - Suppose we have to test function that handles verification of user password. Each user on each computer must have a password. Each password might be atleast 6 to 8 characters long. It might be in capital letters, alphanumeric or in lower case. Then it is just impossible to consider various permutations and combinations for password verification.

31. State the guidelines for debugging.

Following are some debugging guideline suggested by Myers

- Debugging can be effectively carried out by mental analysis.
- If an error cannot be located by yourself, then describe the problem to someone else.
- Use debugging tool only if no alternative is present.
- Avoid experimentation.
- If one bug occurs in one area of code then there are chances of occurrence of more bugs.
- Fix the error and not its symptoms.
- For correcting the existing program, if a new code is added to it then test it rigorously than the original program.
- There are chances of introducing new errors on correction of older ones.
- The process error repair must be conducted in parallel during design phase.
- Change the source code and not the object code/machine code.

Software Implementation Techniques

32. What are the levels at which the testing is done?

Testing can be done in the two levels as given below -

- **Component level testing**: In the component level testing the individual component is tested.
- **System level testing**: In system testing, the testing of group of components integrated to create a system or subsystem is done.

33. What is a critical module and how they are identified?

The module which possess following characteristics is called critical module.

- The module is security related
- The module is located in an untrusted environment
- The module is important to the owner of the software

- The module is not fully trusted

34. How are the software testing results related to the reliability of the software?

During the software testing the program is executed with the intention of finding as much errors as possible. The test cases are designed that are intended to discover yet-undiscovered errors. Thus after testing, majority of serious errors are removed from the system and it is turned into the model that satisfied user requirements.

35. What is smoke testing? [MJ 2017]

- Smoke testing is the initial testing process exercised to check whether the software under test is ready/stable for further testing.
- The term '**Smoke Testing**' is came from the hardware testing, in the hardware testing initial pass is done to check if it did not catch the fire or smoked in the initial switch on.

36. Mention the purpose of stub and driver used for testing [ND 2017]

- Stubs are dummy modules that are always distinguish as "called programs", or you can say that is handle in integration testing (top down approach), it used when sub programs are under construction.
- Stubs are considered as the dummy modules that always simulate the low level modules.
- Drivers are also considered as the form of dummy modules which are always distinguished as "calling programs", that is handled in bottom up integration testing, it is only used when main programs are under construction.
- Drivers are considered as the dummy modules that always simulate the high level modules.

37. List two strategies that address verification. Which types of testing address validation? [MJ 2017]

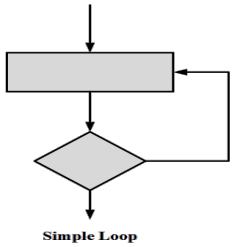
Validation is the process of evaluating the final product to check whether the software meets the business needs. In simple words the test execution which we do in our day to day life are actually the validation activity which includes smoke testing, functional testing, regression testing, systems testing etc…

38. How will you test a simple loop ? [ND 2015]

Simple loops :

The tests can be performed for n number of classes, where

1) n = 0 that means skip the loop completely.
2) n = 1 that means one pass through the loop is tested.
3) n = 2 that means two passes through the loop is tested.
4) n = m that means testing is done when there are m passes where men.
5) Perform the testing when number of passes are n - 1, n, n + 1.



Simple Loop

39. How can refactoring made more effective [MJ 2016]

Refactoring is "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure," according to Martin Fowler, the "father" of refactoring. The concept of refactoring covers practically any revision or cleaning up of source code, but Fowler consolidated many best practices from across the software development industry into a specific list of "refactoring" and described methods to implement them in his book, Refactoring: Improving the Design of Existing Code. While refactoring can be applied to any

programming language, the majority of refactoring current tools have been developed for the Java language

40. Why does software fail after it has passed acceptance testing? [MJ 2016]

- Acceptance criteria may not require the software to be 100% bug-free.
- Test cases may not be able to test for everything that could possibly go wrong. (It is difficult to make things idiot proof because idiots are SO creative!)
- Interactions with new (or unexpected) hardware and third party software might cause the software in question to fail.
- The acceptance tests were not rigorous enough

41. Difference between Alpha Testing and Beta Testing. [MJ 2016]

Differences between Alpha and Beta Testing (Field Testing)

Alpha Testing	Beta Testing (Field Testing)
1. It is always performed by the developers at the software development site.	1. It is always performed by the customers at their own site.
2. Sometimes it is also performed by Independent Testing Team.	2. It is not performed by Independent Testing Team.
3. Alpha Testing is not open to the market and public	3. Beta Testing is always open to the market and public.
4. It is conducted for the software application and project.	4. It is usually conducted for software product.
5. It is always performed in **Virtual Environment**.	5. It is performed in **Real Time Environment**.
6. It is always performed within the organization.	6. It is always performed outside the organization.
7. It is the form of Acceptance Testing.	7. It is also the form of Acceptance Testing.
8. Alpha Testing is definitely performed and carried out at the developing organizations location with the involvement of developers.	8. Beta Testing (field testing) is performed and carried out by users or you can say people at their own locations and site using customer data.
9. It comes under the category of both White Box Testing and Black Box Testing.	9. It is only a kind of Black Box Testing.
10. Alpha Testing is always performed at the time of Acceptance Testing when developers test the product and project to check whether it meets the user requirements or not.	10. Beta Testing is always performed at the time when software product and project are marketed.
11. It is always performed at the developer's premises in the absence of the users.	11. It is always performed at the user's premises in the absence of the development team.
12. Alpha Testing is not known by any other different name.	12 Beta Testing is also known by the name **Field Testing** means it is also known as field testing.
13. It is considered as the User Acceptance Testing (UAT) which is done at developer's area.	13. It is also considered as the User Acceptance Testing (UAT) which is done at customers or users area.

42 What are the testing principles the software engineer must apply while performing the software testing?

- All tests must be traceable to customer requirements.
- Tests should be planned long before testing begins
- Testing should begin in small and progress towards testing in large.
- Exhaustive testing is not possible
- Testing should be done independent third party.

43 Will the exhaustive testing guarantee that the program is 100% correct? A/M 2016

No the exhaustive testing will not guarantee that the program is 100% correct. This is because there are several parameters that affect the correctness of the program.

User acceptance – It is not guaranteed that the program is strictly as per users expectation and will work as per user design.

Performance testing – It is necessary to test whether the program performs to a level that was satisfactory. Similarly the performance should not get degraded if volume load gets increased.

Integration tests – This testing need to be done to check whether the program works properly if it gets integrated with some other software component.

Besides these testing there are many other types of testing that needs to be carried out on the program, and there are chances of a program getting failed in one or more testing.

44 Identify the type of maintenance for each of the following: A/M 2018

a. Correcting the software faults.
b. Adapting the change in environment.

a. Corrective maintenance

b. Adaptive maintenance

45. List the levels of testing. **[MJ 2019]**

There are mainly four testing levels are:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

46. Define reverse engineering **[MJ 2019]**

Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

47. What is test case? **[ND 2019]**

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.

The process of developing test cases can also help find problems in the requirements or design of an application.

48.Define a component .Give example.

A component is a modular, portable, replaceable, and reusable set of well-defined functionality that encapsulates its implementation and exporting it as a higher-level interface.

A component is a software object, intended to interact with other components, encapsulating certain functionality or a set of functionalities. It has an obviously defined interface and conforms to a recommended behavior common to all components within an architecture.

A software component can be defined as a unit of composition with a contractually specified interface and explicit context dependencies only. That is, a software component can be deployed independently and is subject to composition by third parties

PART B Questions Bank

Q.No	Questions	Univ. QP
1	Briefly explain why Software Testing is required and how is it important to producing a quality software product and how it is planned	[MJ 2016]
2	Explain White Box Testing in detail. What are the outcome of White Box testing along with various strategies used to perform White Box testing? [May/June 2015 – 8M]	[MJ 2015] [MJ 2017]
3	What are the steps to be followed to perform Basis Path testing? Explain with an example.	[MJ 2017] [ND 2016]
4	Explain Black Box Testing and related strategies in detail [May/June 2015 – 8M] (or) Explain boundary value Analysis and Equivalence class partitioning	[MJ 2016/2015] [MJ 2017] [ND 2015/2016]
5	Write a short note on different testing strategies adapted in software engineering. [May/June 2015 – 8M]	[MJ 2015]
6	Explain Validation Testing in detail and what are its outcome [Nov/Dec 2013 – 8M]	[ND 2013]
7	What is the need for System testing? Explain various types of system testing in detail.	
8	Differentiate Testing and Debugging activities. Explain the process of debugging in detail. [May/June 2015 – 8M] [May/June 2014 – 8M]	[MJ 2014/2015]
9	Depict the commonly used coding practices that are followed to avoid common errors.	
10	How will you carry out Unit Testing and Integration Testing? What are the needs for various testing strategies?	[MJ 2017] [ND 2015]
11	What are the coding standards that are to be followed by a software developer in order to produce an error free software product? Explain how the developer can adapt to refactoring (8 Marks). (or) Explain how the developer can adapt to refactoring with suitable examples. (16 Marks).	[ND 2014] [MJ/ND 2016]
12	Explain the concept of cyclomatic complexity	[ND 2017]
13	Difference between black box testing and white box testing	[ND 2017/MJ 2019]
14	Explain how the various types of loops are tested	[ND 2017]
15	Difference between Alpha testing and Beta Testing	[MJ 2016]
16	Describe in detail about system testing.	[ND 2014]
17	State the concept of debugging in detail	[MJ 2015]
18	Discuss some of the software implementation Techniques.	[MJ 2018]
19	Write short notes on Refactoring	[MJ 2018/2019]
20	Elaborate in detail the concept of Maintenance and Reengineering	
21	Explain the concept of Business Process Reengineering	[ND 2019]
22	Write Short notes on Reengineering Process Model	
23	Explain Reverse and Forward engineering	

24	List the phases in software reengineering process model and explain each phase.	[MJ 2019]
25	Elaborate path testing and regression testing with an example.	[ND 2019]
26	Outline how the reverse engineering process helps to improve the legacy software	[ND 2019]

University Questions

Part A

2. What is the need for regression testing? **A/M 2015**
3. Write the best practices for "CODING". **A/M 2015**
4. List some of the god coding practices. **N/D 2015**
5. How will you test a simple loop? **N/D 2015**
6. How can refactoring be made more effective? **M/J 2016**
7. Why does software fail after it has passed from acceptance testing? **M/J 2016**
8. **Will the exhaustive testing guarantee that the program is 100% correct? A/M 2016**
9. What methods are used for breaking very long expression and statements? **N/D 2016**
10. What is the difference between verification and validation? Which types of testing address verification? Which types of testing address validation? **N/D 2016**
11. What is smoke testing? **A/M 2017**
12. List two testing strategies that address verification. Which type of testing address validation?
13. Mention the purpose of stub and Driver used for testing. **N/D 2017**
14. Define verification and validation testing. **N/D 2017**
15. What are the testing principle the software engineer must apply while performing the software testing? **A/M 2018**
16. Distinguish between verification and validation. **A/M 2018**
17. Identify the type of maintenance for each of the following: **A/M 2018**
    a. Correcting the software faults.
    b. Adapting the change in environment.

Part B

A/M 2015

1. Describe the various black box and white box testing techniques. Use suitable examples for your explanation. (16)
2. Discuss about the various Integration and debugging strategies followed in software development. (16)

N/D 2015

3. A program spec state the following for an input field: The program shall accept an input value of 4-digit integer equal or greater than 2000 and less than or equal 8000. Determine the test cases using
   i. Equivalence class portioning
   ii. Boundary value analysis.
4. Explain unit testing and integration testing process with an example. (16)

M/J 2016

5. State the need for refactoring. How can a development model benefit by the use of refactoring? (8)
6. Why does software testing need extensive planning? Explain. (8)
7. Compare and contrast alpha and beta testing. (8)
8. Consider a program for determining the previous date. Its input is a triple of day, month and year with the values in the range 1<= month <=12, 1<= day <=31, 1990 <=year <= 2014. The possible outputs would be previous date or invalid input dqate. Design the boundary value test cases.

N/D 2016

9. Consider the pseudocode for simple subtraction given below: (10)
   i. Program 'Simple Subtraction'
   ii. Input (x,y)
   iii. Output (x)
   iv. Output (y)
   v. If x>y then Do
   vi. x-y=z
   vii. Else y – x = z

viii. EndIf
ix. Output (z)
x. Output "End Program"

Perform basis path testing and generate test cases.

10. What is refactoring? When is it needed? Explain with an example. (6)
11. What is black box testing? Explain the different types of black box testing strategies. Explain by considering suitable examples. (16)

A/M2017

12. What is white box testing? Explain. (7)
13. Consider the pseudocode for simple subtraction given below: (6)
    i. Program 'Simple Subtraction'
    ii. Input (x,y)
    iii. Output (x)
    iv. Output (y)
    v. If x>y then Do
    vi. x-y=z
    vii. Else y – x = z
    viii. EndIf
    ix. Output (z)
    x. Output "End Program"

    Perform basis path testing and generate test cases
14. What is integration testing? Discuss any one method in detail. (8)
15. Describe black box testing. Design the bloc-box test suite for the following program. The program computes the intersection point of two straight lines and displays the result. It reads two integer pairs (m1, c1) and (m2, c2) defining the two straight lines of the form y=mx+c. (5)

N/D 2017

16. Consider the following program segment.

```
/*num is the number the function searches in a presorted integer array arr*/
int bin_search(int num)
 {
int min,max;min=0;max=100;
while(min!=max){
if(arr[(min + max)/2]>num)
max=(min+max)/2;
else if(arr[(min + max)/2]
min = (min+max)/2;
else return ((min + max)/2);
}
return(-1);
}
```

    i) Draw the control flow graph for this program segment. **(2)**
    ii) Define cyclomatic complexity. **(2)**
    iii) Determine the cyclomatic complexity for this program. (show the intermediate steps in your computation. Writing only the final result is not sufficient). **(9)**
17. Explain how the various types of loops are tested. (9)
18. Differentiate black box and white box testing. (4)

A/M2018

19. Consider the pseudocode for simple subtraction given below: (9)
    a. Program 'Simple Subtraction'
    b. Input (x,y)
    c. Output (x)
    d. Output (y)
    e. If x>y then Do
    f. x-y=z
    g. Else y – x = z
    h. EndIf

i. Output (z)
j. Output “End Program”

Perform basis path testing and generate test cases

20. Explain top down integration testing with an example. (4)
21. Write notes on: (13)
    i. Regression testing
    ii. Refactoring
    iii. Debugging