1. Software Testing

**Introduction:**

Software Engineering is a discipline whose aim is the production of quality software, that satisfies the customer’s needs, Product delivered on time and within budget. The Goal of software engineering is to produce software product, that is economic, safety, errorless and useful for people. Software testing is one of the important phase of software development life cycle. This phase ensure, the product is safely ready for generalization to fill users requirement and ready to fill society needs.

Testing is an endless process, time taken and expensive. Therefore several questions arises: How much time are required for testing a product? Which testing techniques should be used for better performance of software product? In how many test cases we can find maximum number of bugs/faults? Which test case should be execute first? Which
test should be first. What methodology/algorithm should be used to find bugs as early as possible.

2. Testing approaches of software testing

2.1 Functional Testing: It consists of User Interface Testing, Validation Testing, functionality Testing


2.2 Integration: It consists of Intra-Modular Testing, Application, and System Testing, Integration Testing

2.3 Test case: It is a set of test data and test programs (test scripts) and their expected results. A test case validates one or more system requirements and generates a pass or fail.

2.4 Test suite: It is a collection of test scenarios and/or test cases that are related or that may cooperate with each other. In other words Test suit contains set of test cases, which are both related or unrelated.

2.5 Test scenario: It is set of test cases that ensure that the business process flows are tested from end to end. They may be independent tests or a series of tests that follow each other, each dependent on the output of the previous one. The terms “test scenario“ and test case” are often synonymously.

2.6 Regression Testing: Regression testing means rerunning existing tests against the modified code to determine whether the changes break anything that worked prior to the change
and by writing new tests where necessary. In other word Regression Testing means that test cases from existing test suites to build confidence that software changes have no unintended side-effects. The ideal process would be to create an extensive test suite and tin it after each and every change.

Select data for test case design and execution from academic and industrial environment. Define all possible test cases for selected test data problem. Analysis the all possible test suits under regression testing. Rearranging the test cases in test suit for finding the software problem with minimum test case and which takes minimum time to execute test case. Design methodology, which is applicable on different software application / software product for batter solution under regression testing with minimum time and minimum cost.

Regression testing is an expensive process. Regression testing is compulsory for proper functioning of software product as per current market trend. It is an expensive, time consuming
endless process in the software development life cycle. A test case prioritization, assigns each test case a selection probability according to its potential ability to achieve some certain testing goal. It selects prioritized test cases to run to get higher testing performance [6].

Test case prioritization techniques could be of great benefit to increasing the effectiveness of test suites in practice. Test case prioritization is a technique. It helps to increase the rate of fault detection or code. Whereas Regression test prioritization is often performed in a time constrained execution environment in which testing only occurs for a fixed time period. Some time it is complex and time consuming process due to there may be insufficient resources to allow for the re-execution of all test cases. On this stage test case prioritization techniques are used to handle the situation, which is aim to improve the effectiveness of regression testing by ordering the test cases so that the most beneficial are executed first. Several test case prioritization techniques are
derived to schedule the test case on priority basis for their minimum time execution, minimum cost or minimum weight. Test case prioritization techniques schedule test cases for regression testing in an order that increases their ability to meet some performance goal. One performance goal, rate of fault detection, measures how quickly faults are detected within the testing process.

APFD (Average of percentage of fault detected)[1] is derived and measures by Gregg Rothermel. In continuation Sebastian Elbaum et al 2001 have created such a metric by adapting APFD metric called “cost-cognizant” metric APFDc[2], APFD (Average percentage of fault detection) was base on simple fault detection with equal severity and cost of test case. Where APFDc described based on cost of test case and severity of faults.
2.7 Testing Types and Techniques

Testing types

Testing types refer to different approaches towards testing a computer program, system or product. The two types of testing are black box testing and white box testing,

2.7.1 Testing Techniques

Testing techniques refer to different methods of testing particular features a computer program, system or product. Each testing type has its own testing techniques while some techniques combine the feature of both types. Some techniques are

- Error and anomaly detection technique
- Interface checking
- Physical units checking
- Loop testing (Discussed in detail in this chapter)
• Basis Path testing/McCabe’s cyclomatic number (Discussed in detail in this chapter)

• Control structure testing (Discussed in detail in this chapter)

• Error Guessing (Discussed in detail in this chapter)

• Boundary Value analysis (Discussed in detail in this chapter)

• Graph based testing (Discussed in detail in this chapter)

• Equivalence partitioning (Discussed in detail in this chapter)

• Instrumentation based testing

• Random testing

• Domain testing

• Halstead’s software science

• And many more
Some of these and many others would be discussed in the later sections of this chapter.

2.7.2 White Box Testing

White box testing involves looking at the structure of the code. When we know the internal structure of a product, tests can be conducted to ensure that the internal operations performed according to the specification. And all internal components have been adequately exercised. In other word WBT tends to involve the coverage of the specification in the code.

Code coverage is defined in six types as listed below.

- **Segment coverage** – Each segment of code b/w control structure is executed at least once.

- **Branch Coverage or Node Testing** – Each branch in the code is taken in each possible direction at least once.

- **Compound Condition Coverage** – When there are multiple conditions, we must test not only each direction but also
each possible combinations of conditions, which is usually done by using a ‘Truth Table’

- Basis Path Testing – Each independent path through the code is taken in a pre-determined order. This point will further be discussed in other section.

- Data Flow Testing (DFT) – In this approach track the specific variables through each possible calculation, thus defining the set of intermediate paths through the code i.e., those based on each piece of code chosen to be tracked. Even though the paths are considered independent, dependencies across multiple paths are not really tested for by this approach. DFT tends to reflect dependencies but it is mainly through sequences of data manipulation. This approach tends to uncover bugs like variables used but not initialize, or declared but not used, and so on.
• Path Testing – Path testing is where all possible paths through the code are defined and covered. This testing is extremely laborious and time consuming.

• Loop Testing – In addition top above measures, there are testing strategies based on loop testing. These strategies relate to testing single loops, concatenated loops, and nested loops. Loops are fairly simple to test unless dependencies exist among the loop or b/w a loop and the code it contains.

2.7.2.1 What do we do in White Box Testing?

In WBT, we use the control structure of the procedural design to derive test cases. Using WBT methods a tester can derive the test cases that

• Guarantee that all independent paths within a module have been exercised at least once.
- Exercise all logical decisions on their true and false values.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to ensure their validity.

White box testing (WBT) is also called *Structural* or *Glass box* testing.

**2.7.2.2 Tools used for White Box testing:**

Few Test automation tool vendors offer white box testing tools which:

1) Provide run-time error and memory leak detection;

2) Record the exact amount of time the application spends in any given block of code for the purpose of finding inefficient code bottlenecks; and

3) Pinpoint areas of the application that have and have not been executed.
**Basis Path Testing**

Basis path testing is a *white box testing* technique first proposed by Tom McCabe. The Basis path method enables to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test Cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

**Flow Graph Notation**

The flow graph depicts logical control flow using a diagrammatic notation. Each structured construct has a corresponding flow graph symbol.

**Cyclomatic Complexity**

Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program. When used in the context of a basis path testing method, the value computed for Cyclomatic complexity defines the number for independent paths in the basis set of a program and
provides us an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

**Computing Cyclomatic Complexity**

Cyclomatic complexity has a foundation in graph theory and provides us with extremely useful software metric. Complexity is computed in one of the three ways:

1. The number of regions of the flow graph corresponds to the Cyclomatic complexity.

2. Cyclomatic complexity, \( V(G) \), for a flow graph, \( G \) is defined as

\[
V \ (G) = E - N + 2
\]

Where \( E \), is the number of flow graph edges, \( N \) is the number of flow graph nodes.
3. Cyclomatic complexity, $V(G)$ for a flow graph, $G$ is also defined as:

$$V(G) = P + 1$$

Where $P$ is the number of predicate nodes contained in the flow graph $G$.

**Graph Matrices**

The procedure for deriving the flow graph and even determining a set of basis paths is amenable to mechanization. To develop a software tool that assists in basis path testing, a data structure, called a graph matrix can be quite useful.

A Graph Matrix is a square matrix whose size is equal to the number of nodes on the flow graph. Each row and column corresponds to an identified node, and matrix entries correspond to connections between nodes.
**Limitations**

Unfortunately in White Box Testing, exhaustive testing of a code presents certain logistical problems. Even for small programs, the number of possible logical paths can be very large. For instance, a 100 line C Language program that contains two nested loops executing 1 to 20 times depending upon some initial input after some basic data declaration. Inside the interior loop four if-then-else constructs are required. Then there are approximately $10^{14}$ logical paths that are to be exercised to test the program exhaustively. Which means that a magic test processor developing a single test case, execute it and evaluate results in one millisecond would require 3170 years working continuously for this exhaustive testing which is certainly impractical. Exhaustive WBT is impossible for large software systems. But that doesn’t mean WBT should be considered as impractical. Limited WBT in which a limited no. of important logical paths are selected and exercised
and important data structures are probed for validity, is both practical and WBT. It is suggested that white and black box testing techniques can be coupled to provide an approach that validates the software interface selectively ensuring the correction of internal working of the software.

2.7.3 Black Box Testing

Black box is a test design method. Black box testing treats the system as a "black-box", so it doesn't explicitly use knowledge of the internal structure. Or in other words the Test engineer need not know the internal working of the “Black box”. It focuses on the functionality part of the module.

Some people like to call black box testing as behavioral, functional, opaque-box, and closed-box. While the term black box is most popularly use, many people prefer the terms "behavioral" and "structural" for black box and white box respectively. Behavioral test design is slightly different from
black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged.

Personally we feel that there is a tradeoff between the approaches used to test a product using white box and black box types.

There are some bugs that cannot be found using only black box or only white box. If the test cases are extensive and the test inputs are also from a large sample space then it is always possible to find majority of the bugs through black box testing.

**Advantages of Black Box Testing**

- Tester can be non-technical.

- This testing is most likely to find those bugs as the user would find.

- Testing helps to identify the vagueness and contradiction in functional specifications.

- Test cases can be designed as soon as the functional specifications are complete
**Disadvantages of Black Box Testing**

- Chances of having repetition of tests that are already done by programmer.

- The test inputs needs to be from large sample space.

- It is difficult to identify all possible inputs in limited testing time. So writing test cases is slow and difficult

Chances of having unidentified paths during this testing

**11.1 Unit Testing**

This is a typical scenario of Manual Unit Testing activity-

A Unit is allocated to a Programmer for programming. Programmer has to use ‘Functional Specifications’ document as input for his work.

Programmer prepares ‘Program Specifications’ for his Unit from the Functional Specifications. Program Specifications describe the programming approach, coding tips for the Unit’s coding.
Using these ‘Program specifications’ as input, Programmer prepares ‘Unit Test Cases’ document for that particular Unit. A ‘Unit Test Cases Checklist’ may be used to check the completeness of Unit Test Cases document.

‘Program Specifications’ and ‘Unit Test Cases’ are reviewed and approved by Quality Assurance Analyst or by peer programmer.

The programmer implements some functionality for the system to be developed. The same is tested by referring the unit test cases. While testing that functionality if any defects have been found, they are recorded using the defect logging tool whichever is applicable. The programmer fixes the bugs found and tests the same for any errors.

3. Review of Literature

3.1 G. Rothermel, et al 1999, describe that Test case prioritization techniques schedule test cases for execution in an order that attempts to maximize some objective function.
They had discussed nine different prioritization techniques:
(a) No prioritization (b) Random prioritization (c) Optimal prioritization (d) Total branch coverage prioritization. (e) Additional branch coverage prioritization (f) Total fault-exposing-potential (FEP) prioritization. (g) Additional fault-exposing-potential (FEP) prioritization. (h) Total statement coverage prioritization (i) Additional statement coverage prioritization.

They have used a weighted average of the percentage of faults detected, or APFD on a test suite of five different test cases. Arranged test cases in different sequence to find out percentage of maximum fault detection. According to them Test case prioritization techniques schedule test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal.

3.2 G. Rotermel and Harrold M.J. [2] had described Analyzing Regression Test Selection Techniques, They had proposed,
Regression testing is a necessary but expensive maintenance activity aimed at showing that code has not been adversely affected by changes. Regression test selection techniques reuse tests from an existing test suite to test a modified program. Many regression test selection techniques have been proposed; however, it is difficult to compare and evaluate these techniques because they have different goals. This paper outlines the issues relevant to regression test selection techniques, and uses these issues as the basis for a framework within which to evaluate the techniques.

3.3 G. Rotermel and Harrold M.J. [3] had described Efficient Regression Test Selection Technique; Regression testing is an expensive but necessary maintenance activity performed on modified software to provide confidence that changes are correct and do not adversely affect other portions of the software. A regression test selection technique chooses, from
an existing test set, tests that are deemed necessary to validate modified software. We present a new technique for regression test selection. Our algorithms construct control flow graphs for a procedure or program and its modified version, and use these graphs to select tests that execute changed code from the original test suite. We prove that under certain conditions, the set of tests our technique selects includes every test from the original test suite that can expose faults in the modified procedure or program. Under these conditions our algorithms are safe. Moreover, although our algorithms may select some tests that cannot expose faults, they are at least as precise as other safe regression test selection algorithms, unlike many other regression test selection have implemented our algorithms. Initial empirical studies indicate that our technique can significantly reduce the cost of regression testing modified software.

3.4 Sebastian Elbaum et al, [6] describe in his research that Rothermel et al. [04] distinguishes two types of test case
prioritization: general and version-specific. In general test case prioritization, given program P and test suite T, test cases in T are prioritized with the goal of finding a test case order that will be useful over a sequence of subsequent modified versions of P. In contrast, in version-specific test case prioritization, given program P and test suite T, test cases in T are prioritized with the intent of finding an ordering that will be useful on a specific version P0 of P. Version specific prioritization is performed after a set of changes have been made to P and prior to regression testing P0.

3.5 Zheng Li, Mark Harman, et al [9], they have described Search Algorithms for Regression Test Case Prioritization and results from an empirical study of the application of several greedy, meta heuristic, and evolutionary search algorithms to six programs, ranging from 374 to 11,148 lines of code for three choices of fitness metric. They addresses the problems of choice of fitness metric, characterization of landscape
modality, and determination of the most suitable search technique to apply. The empirical results replicate previous results concerning Greedy Algorithms.

3.6 R. C. Bryce [11], Event-driven software (EDS) is a widely used class of software that takes sequences of events as input, changes state, and outputs new event sequences. Event-driven software (EDS) is a class of software that is quickly becoming ubiquitous. All EDS share a common event-driven model – they take sequences of events (i.e., messages, mouse-clicks) as input, change their state, and (sometimes) output an event sequence. Extend software interaction testing. Software interaction testing can systematically examine event interactions. Further, test suites can be generated with a logarithmic guarantee on size [10]. For the purpose of testing GUIs we need to extend “software interaction testing” to consider sequences of events (that contain event interactions).
3.7 Hyunsook Do et al., Regression testing is an expensive process used to validate modified software. Test case prioritization techniques improve the cost effectiveness of regression testing by ordering test cases such that those that are more important are run earlier in the testing process. Test case prioritization techniques (e.g., [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]) offer an alternative approach to improving regression testing cost-effectiveness. Prioritization can improve cost effectiveness in two ways. First, prioritization can help engineers reveal faults early in testing, allowing them to begin debugging activities earlier in the testing cycle than might otherwise be possible. Second, in the case in which testing activities are cut short and test cases must be omitted, prioritization can improve the chances that important test cases will have been executed. In this case, cost savings related to early fault detection (by those test cases that are executed) still apply, and additional benefits accrue from lowering the number of faults that might otherwise be missed through less appropriate runs of partial test suites.
3.8 Pavan Kumar Chittimalli & Mary Jean Harrold [21] their work consists of an RTS (Regression Test Selection) technique, adapted from existing RTS techniques, for use with system requirements. Our new technique provides a way to use system requirements and their associations with test cases to select the important test cases for rerunning on the changed versions of the system.

3.9 Lijun Mei, Zhenyu Zhan et al [22] test case prioritization [19] is important in regression testing [24][25]. It schedules the test cases in a regression test suite with a view to maximizing certain objectives (such as revealing faults earlier), which help reduce the time and cost required to maintain service-oriented business applications. Existing regression testing techniques for such applications focus on testing individual services [26] or workflow programs [23]. Surprisingly, to the best of our knowledge, the integration complexity raised by non-
imperative artifacts such as XPath and WSDL (Web Service Description Language) [27] among workflow steps has been inadequately addressed in regression testing research. They have propose a multilevel coverage model to capture the coverage requirements of these artifacts. Level 1 covers only the workflow, level 2 covers both workflow and XPath, and level 3 covers workflow, XPath, and WSDL. Through the level-by-level use of coverage data for test cases, we propose a new family of test case prioritization techniques.

4. Methodology of the Research Work:

Software testing is a phase of software development life cycle. The objective of software testing is to show incorrectness and when error is detected testing is considered to success. The purpose of software testing is to detect errors /faults/bugs, not to correct them in the software. Aim of software testing is to produce a software product, that is economic, errorless and
useful and safe for people. Goal of software testing is to find bugs and fixed them as early as possible.

Test case is a set of test data and test programs (test scripts) and their expected results.

Test suite is a collection of test scenarios and/or test cases that are related or that may cooperate with each other. In other words Test suit contains set of test cases, which are both related or unrelated. Test scenario is set of test cases that ensure that the business process flows are tested from end to end. Regression Testing means rerunning existing tests against the modified code to determine whether the changes break anything that worked prior to the change and by writing new tests where necessary. In other word Regression Testing means that test cases from existing test suites to build confidence that software changes have no unintended side-effects.

Test Case Prioritization Techniques, It provides another method for assisting with regression testing. These techniques
let testers order their test cases so that those test cases with the highest priority, according to some criterion, are executed earlier in the regression testing process than lower priority test cases. For example: Testers might wish to schedule test cases in an order that achieves code coverage at the fastest rate possible, exercises features in order of expected frequency of use, or exercises subsystems in an order that reflects their historically demonstrated propensity to fail.

Goal of Test Case Prioritization: Test case prioritization schedule test cases in order to increase their ability to meet some performance goal: Rate of fault detection Rate of code coverage Rate of increase of confidence in reliability Rate of fault detection is one of the goal to measure - How quickly faults are detected within the testing process.

**Test Case Prioritization**

**Definition:**

The Test Case Prioritization Problem:
Given: T is a test suite; PT is the set of permutations of T; (all possible prioritizations of T) f is a function from PT to a real number (award value) then Find T' ∈ PT such that (for all T'') T'' ∈ PT, (T'' != T') [f (T') >= f (T'')]

5. Objectives:

1. Survey of software testing techniques in software engineering

2. Study of different methodology test case prioritization under Regression Testing in Software Testing

3. To Design and develop a methodology prioritization of test case under Regression Testing.

4. Comparison results
Schedule of Activity

<table>
<thead>
<tr>
<th>Activity Steps</th>
<th>Activity Name</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>Survey of work</td>
<td>0-6 Months</td>
</tr>
<tr>
<td>Second</td>
<td>Design and develop methodology</td>
<td>7-12 Months</td>
</tr>
<tr>
<td></td>
<td>&amp; publication in Journals</td>
<td></td>
</tr>
<tr>
<td>Third</td>
<td>comparison of my results</td>
<td>13-18 Months</td>
</tr>
<tr>
<td>Forth</td>
<td>Submission of Thesis</td>
<td>18-21 Months</td>
</tr>
</tbody>
</table>
Proposed Chapter Scheme:

1. Introduction to software testing in Software Engineering

2. Survey of Software testing methodologies and testing techniques

3. Prioritization techniques of test case in regression testing

4. Design and develop methods for test case prioritization in regression testing

5. Conclusion

6. Bibliography