Role of Testing in Phases of SDLC and Quality

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science & Engineering

By:
Youddha Beer Singh
(80732027)

Under the supervision of:
Mrs. Shivani Goel
Lecturer

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004
JUNE – 2009

CERTIFICATE

I hereby certify that the work which is being presented in the thesis report entitled, **"Role of Testing In Phases of SDLC and Quality",** submitted by me in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mrs. Shivani Goel* and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(***Youddha Beer Singh***)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mrs. Shivani Goel)
Lecturer
Computer Science and Engineering Department
Thapar University, Patiala

Countersigned by:

(Dr. Rajesh Kumar Bhatia)
Assistant Professor & Head,
Computer Science & Engineering Department
Thapar University, Patiala

(Dr. R.K. Sharma)
Dean (Academic Affairs)
Thapar University, Patiala

ACKNOWLEDGEMENT

No volume of words is enough to express my gratitude towards my guide, Mrs. Shivani Goel, Lecturer, Computer Science and Engineering Department, Thapar University, who has been very concerned and has aided for all the material essential for the preparation of this thesis report. She has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research-oriented venture.

I am also thankful to Dr. Rajesh Kumar Bhatia, Head of Department, CSED and Mrs. Inderveer Channa, P.G. Coordinator, for the motivation and inspiration that triggered me for this thesis work.

I would also like to thank the staff members and my colleagues who were always there in the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis. Most importantly, I would like to thank my parents and the Almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Singh

Youddha Beer Singh

(80732027)

ABSTRACT

Software testing is a technique aimed at evaluating an attribute or capability/usability of a program or product/system and determining that it meets its quality. Although crucial to software quality and widely deployed by programmer & testers, software testing still remains an art, due to limited understanding of the principles of software. The difficulty in software testing stem from the complexity of software we cannot completely test a program with moderate complexity. Testing is more than just debugging. The purpose of testing can be quality assurance, verification and validation, or reliability estimation. Software testing is a trade off between budget, time and quality. There is various type of testing, We choose that type of testing technique which is suitable according to our quality attribute. We are concerned with the life cycle of finding the debug by which we can easily understand that how the debugs are found and report it. Quality is the central concern of software engineering. Testing is the single most widely used approach to ensuring software quality.

This thesis report presents various types of software testing techniques and their classification. In this report a model "software development life cycle testing model" is proposed in which we categorise all type of testing techniques related to quality attribute and to test all phases of SDLC and identify that which type of testing technique can be applied to which type of SDLC phase.

Software testing is an important technique for assessing the quality of a software product. In this thesis, various types of software testing technique and various attributes of software quality are explained. Identifying the types of testing that can be applied for checking a particular quality attribute is the aim of this thesis report. All types of testing can not be applied in all phases of software development life cycle. Which testing types are applicable in which phases of life cycle of software development is also summarized.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

Chapter 1: INTRODUCTION

1.1 Software Testing

Software testing is both a discipline and a process. It is a separate discipline from software development. Software development is the process of coding functionality to meet defined end-user needs. While Software testing tends to be considered a part of development, it is really its own discipline and should be tracked as its own project. Software testing, while working very closely with development, should be independent enough to be able to hold-up or slow product delivery if quality objectives are not met [21]. The iterative process of software testing consists of

- Designing tests
- Executing tests
- Identifying problems
- Getting problems fixed

1.1.1 Objective of Testing

The objective of software testing is to find problems and fix them to improve quality. Software testing typically represents 40% of a software development budget. There are four main objectives of testing [1]:

1. **Demonstration:** It show that the system can be used with acceptable risk, demonstrate functions under special conditions and show that products are ready for integration or use.

2. **Detection:** It discovers defects, errors, and deficiencies. Determine system capabilities and limitations quality of components, work products, and the system.

3. **Prevention:** It provides information to prevent or reduce the number of errors clarify system specifications and performance. Identify ways to avoid risks and problems in the future.

4. **Improving quality:** By doing effective testing, we can minimize errors and hence improve the quality of software.

1.1.2 Need for testing

Well, while making food, it’s ok to have something extra, people might understand and eat the things we made and may well appreciate our work. But this isn't the case with software project development [6]. If we fail to deliver a reliable, good and problem free software solution, we fail in our project and probably we may loose our client. So in order to make it sure, that we provide our client a proper software solution, we go for testing. We check out if there is any problem, any error in the system, which can make software unusable by the client. We make software testers test the system and help in finding out the bugs in the system to fix them on time [9]. We find out the problems and fix them and again try to find out all the potential problems

1.1.3 Phases in Software Testing

Although many test teams use test tools or scripts to automate testing activities, there's a lot about testing which is just simply labour intensive [4]. Here are just some of the activities involved:

- **Planning and developing test cases**: writing test plans and documentation, prioritizing the testing based on assessing the risks, setting up test data, organising test teams [4].

- **Setting up the test environment:** an application will be tested using multiple combinations of hardware and software and under different conditions. Also, setting up the prerequisites for the test cases themselves [4].

- **Writing test harnesses and scripts:** developing test applications to call the API directly in order to automate the test cases. Writing scripts to simulate user interactions [5].

- **Planning, writing and running load tests:** non-functional tests to monitor an application's scalability and performance. Looking at how an application behaves under the stress of a large number of users [6].

- **Writing bug reports:** communicating the exact steps required to reproduce unexpected behaviour on a particular configuration. Reporting to development team with test results [4].

1.1.4 Test Planning and Process

To ensure effective testing proper test planning is important an effective testing process will comprise of the following steps [1]:

- Test Strategy and Planning
- Review Test Strategy to ensure its aligned with the Project Goals
- Design/Write Test Cases
- Review Test Cases to ensure proper Test Coverage
- Execute Test Cases
- Capture Test Results
- Track Defects
- Capture Relevant Metrics
- Analyze

The testing process and the test cases should cover:

- All the scenarios that can occur when using the software application.
- Each business requirement that was defined for the project.
- Specific levels of testing should cover every line of code written for the application.

1.1.5 Testing and Debugging

The purpose of debugging is to locate and fix the offending code responsible for a symptom violating a known specification. Debugging typically happens during three activities in software development, and the level of granularity of the analysis required for locating the defect differs in these three. The first is during the coding process, when the programmer translates the design into an executable code. During this process the errors made by the programmer in writing the code can lead to defects that need to be quickly detected and fixed before the code goes to the next stages of development. Most often, the developer also performs unit testing to expose any defects at the module or component level. The second place for debugging is during the later stages of testing, involving multiple components or a complete system, when unexpected behaviour such as wrong return codes or abnormal program termination

may be found. A certain amount of debugging of the test execution is necessary to conclude that the program under test is the cause of the unexpected behaviour.

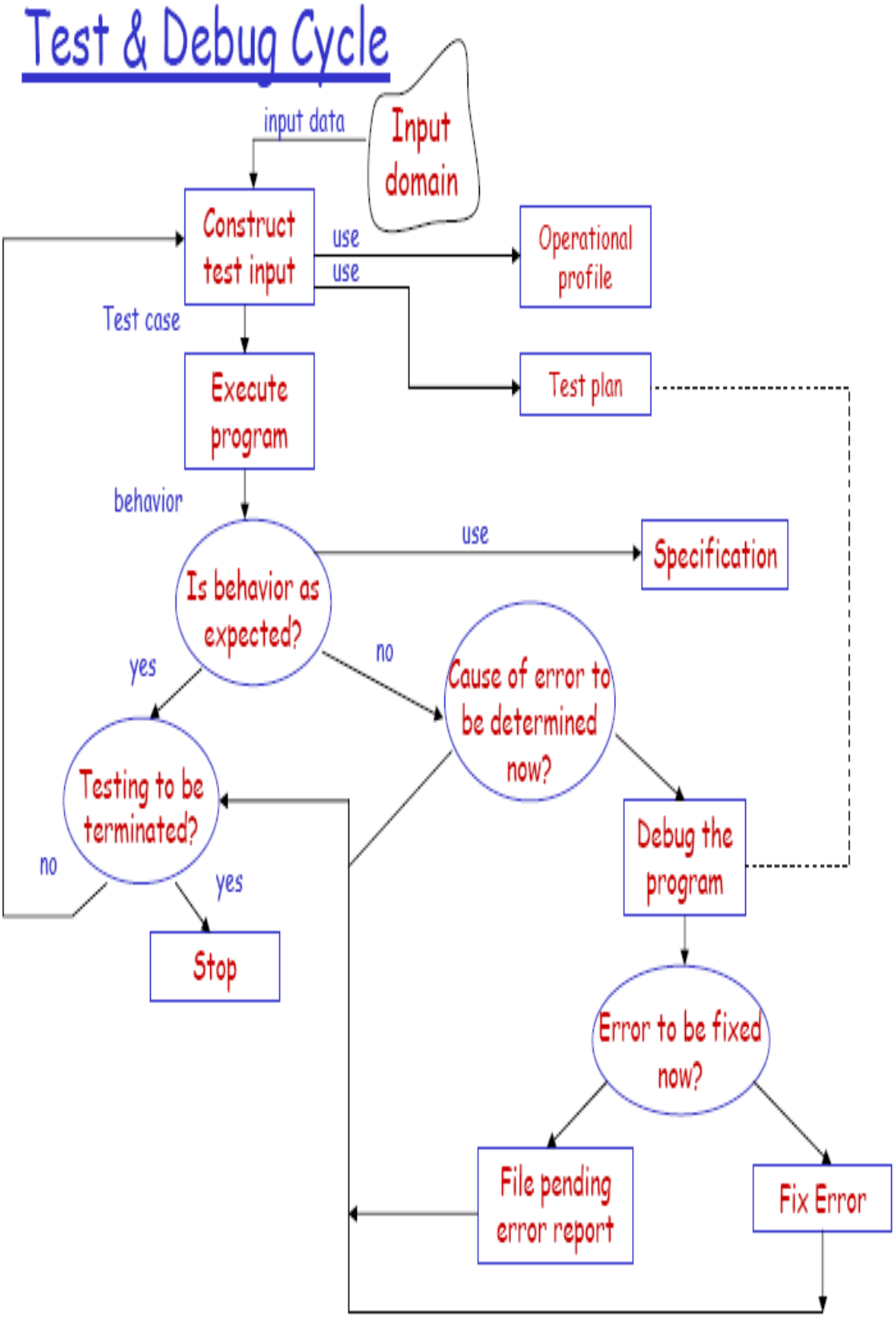


Fig 1.1: Test and debug cycle [12]

1.2 Software quality

There are two generally accepted meanings of quality [10]. The first is that quality means "meeting requirements." With this definition, to have a quality product, the requirements must be measurable, and the product's requirements will either be met or not met. The second is, the quality definition by the customer “whether the product or service does what the customer needs”. Another way of wording it is "fit for use." There should also be a description of the purpose of the product, typically documented in a customer "requirements specification". The requirements are the most important document, and the quality system revolves around it. In addition, quality attributes are described in the customer's requirements specification. Examples include usability, the relative ease with which a user communicates with the application portability, the capability of the system to be executed across a diverse range of hardware architectures; and reusability, the ability to transfer software components constructed in one software system into another [1]. We can classify all quality features into two categories based on the point where they can be applied i.e. on code or application.

Static attributes: Static attributes refer to the actual code (maintainable and testable code) and related documentation (Correct and complete documentation).

Dynamic attributes: Dynamic attributes refer to the behaviour of the application while in use reliability, correctness, completeness, consistency, usability and performance.

1.3 SDLC Models

Software development life cycle is basically a systematic way of developing software. It includes various phases starting from the functional requirement of software (means what software is supposed to do). After that designing takes place then development and then testing. After testing is finished, the source code is generally released for Unit Acceptance Testing (UAT) in client testing environment. After approval from client, the source code is released into production environment [1]. There is various software development approaches defined and designed which are used during development process of software, these approaches are also referred as "Software Development Process Models". Each process model follows a particular life cycle in

order to ensure success in process of software development. Various models of SDLC are:

- Water fall model
- Proto type model
- Spiral model
- Iterative model
- RAD model
- V-model

This chapter summarised three topics, testing technique, SDLC and software quality in second chapter describes various types of testing technique, third chapter describes various SDLC model and software quality attribute, fourth chapter describes problem statement and fifth chapter describes the purposed solution.

Chapter 2: SOFTWARE TESTING

2.1 Software Testing Types

Many different testing techniques have been invented and most of the tests that are generally performed are discussed. Each testing technique serves a different purpose for testing different artifacts like design, code, plan SRS, etc. All the testing technique categorised in black box, white box or grey box testing. The technique which test external behaviour of the system are categorised in black box testing and which test internal behaviour are called white box testing and which test both internal and external are called grey box testing.

2.1.1 Acceptance Testing

Acceptance tests are created from user requirements. During an iteration the user stories selected during the iteration planning meeting will be translated into acceptance tests. The customer specifies scenarios to test when a user story has been correctly implemented. Acceptance tests are a form of black box testing. Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority. Acceptance tests are also used as regression tests prior to a production release. A user story is not considered complete until it has passed its acceptance tests [22]. This means that new acceptance tests must be created each iteration or the development team will report zero progress. A principal purpose of acceptance testing is that, once completed successfully, and provided certain additional (contractually agreed) acceptance criteria are met, the sponsors will then sign off on the system as satisfying the contract (previously agreed between sponsor and manufacturer), and deliver final payment.

2.1.2 Ad hoc Testing

Ad hoc testing is a term commonly used for the tests carried out without planning software and documentation. The tests are intended to be executed only once, unless a defect is discovered. This is part of the exploratory test, which is the least formal of test methods. In this context, it has been criticized because it is not structured, but it

can also be a strength the important bugs can be found quickly. It is performed with improvisation; the tester tries to find bugs with all means that seem appropriate. This test is most often used as a complement to other types of tests such as regression tests [3]. We affirm that the ad hoc test is a special case of Exploratory Testing. During the exploration testing, we will find a large number of ad hoc tests cases (one-off tests), but some will not. One way to distinguish between the two is to examine the notes associated with an exploratory test. In general, exploratory tests have little or no formal documentation, but the result and more notes. If the notes are detailed enough that the test can be repeated by reading, it is less likely to be an ad hoc test. Conversely, if there is no note for an exploratory test, or if the notes are intended to guide the efforts of more tests to reproduce the test, then it is almost certainly an ad hoc test.

2.1.3 Agile Testing

Agile software development is a conceptual framework for software engineering that promotes development iterations throughout the life-cycle of the project. There are many agile development methods, Most minimize risk by developing software in short amounts of time. Software developed during one unit of time is referred to as an iteration, which may last from one to four weeks. Each iteration is an entire software project: including planning, requirements analysis, design, coding, testing, and documentation An iteration may not add enough functionality to warrant releasing the product to market but the goal is to have an available release (without bugs) at the end of each iteration [3]. At the end of each iteration, the team re-evaluates project priorities. Agile methods also emphasize working software as the primary measure of progress. Combined with the preference for face-to-face communication, agile methods produce very little written documentation relative to other methods. This has resulted in criticism of agile methods as being undisciplined.

2.1.4 Alpha Testing

Alpha testing takes place at developers' sites, and involves testing of the operational system by internal staff, before it is released to external customers. Testing of an application when development is near completion, Minor design changes may still be made as a result of such testing [22]. Typically done by end-users or others, but not by programmers or testers.

2.1.5 Basis Path Testing

This testing mechanism was proposed by McCabe. Its aim is to derive a logical complexity measure of a procedural design and use this as a guide for defining a basic set of execution paths [22]. Test cases which exercise basic set will execute every statement at least once identifying tests based on flow and paths of a program or system.

A basis set is a set of linearly independent paths that can be used to construct any path through the program flow graph. A path can be associated with a vector, where each element in the vector is the number of times that an edge is traversed. For example, consider a graph with 4 edges: a, b, c and d. The path ac can be represented by the vector [1 0 1 0]. Paths are combined by adding or subtracting the paths' vector representations. Each path in the basis set can not be formed as a combination of other paths in the basis set [4]. Also, any path through the control flow graph can be formed as a combination of paths in the basis set.

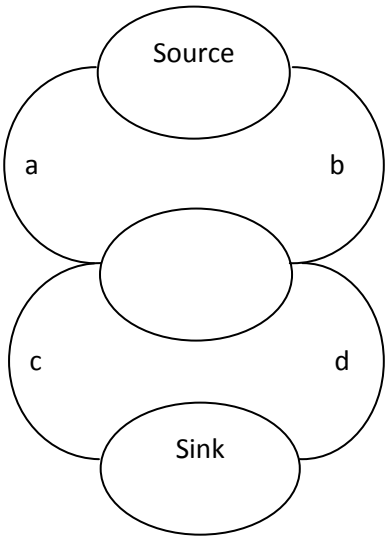


Fig 2.1: Flow graph

Fig 2.1 shows a simplified control flow graph. While a complete flow graph would not have two edges going to the same destination, this requirement has been relaxed to keep the number of paths to a manageable size for this example. A basis set for this graph is {ac, ad, bc}. The path bd can be constructed by the combination bc + ad – ac as shown in this table.

Table 2.1: Basis Path Table

Edge	bd	bc	bc+ad	bc+ad-ac
a	0	0	1	0
b	1	1	1	1
c	0	1	1	0
d	1	0	1	1

2.1.6 Beta Testing

Beta testing takes place at customers' sites, and involves testing by a group of customers who use the system at their own locations and provide feedback, before the system is released to other customers Testing is done when development and testing are essentially completed and to find final bugs and problems before final release [4]. It is typically done by end-users or others, not by programmers or testers.

2.1.7 Black Box Testing

Black box testing takes an external perspective of the test object to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid input and determines the correct output. There is no knowledge of the test object's internal structure. This method of test design is applicable to all levels of software testing: unit, integration, functional testing, system and acceptance [1]. The higher the level, and hence the bigger and more complex the box, the more one is forced to use black box testing to simplify. While this method can uncover unimplemented parts of the specification, one cannot be sure that all existent paths are tested.

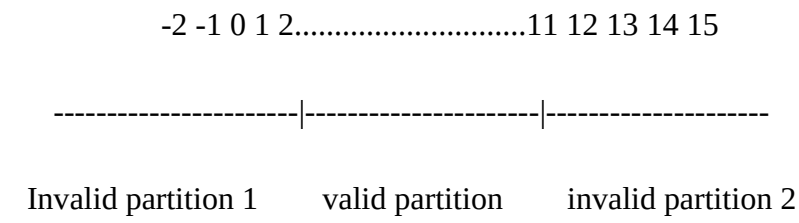
2.1.8 Bottom-Up Testing

Bottom-up testing is an approach to integration testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested. A bottom-up approach is piecing together systems to give rise to grander systems, thus making the original systems sub-systems of the emergent system. In a bottom-up approach the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which

then in turn are linked, sometimes in many levels, until a complete top-level system is formed [1]. This strategy often resembles a "seed" model, whereby the beginnings are small but eventually grow in complexity and completeness. However, “organic strategies” may result in a tangle of elements and subsystems, developed in isolation and subject to local optimization as opposed to meeting a global purpose.

2.1.9 Boundary Value Testing

The expected input and output values should be extracted from the component specification. The input and output values to the software component are then grouped into sets with identifiable boundaries. Each set, or partition, contains values that are expected to be processed by the component in the same way. Partitioning of test data ranges is done in the test case design technique. It is important to consider both valid and invalid partitions when designing test cases. For an example [19] where the input values were months of the year expressed as integers, the input parameter 'month' might have the following partitions:



The boundaries are the values on and around the beginning and end of a partition. If possible test cases should be created to generate inputs or outputs that will fall on and to either side of each boundary this would result in three cases per boundary. The test cases on each side of a boundary should be in the smallest increment possible for the component under test. In the example above there are boundary values at 0,1,2 and 11,12,13. If the input values were defined as decimal data type with 2 decimal places then the smallest increment would be the 0.01. Where a boundary value falls within the invalid partition the test case is designed to ensure the software component handles the value in a controlled manner. [4]. Boundary value analysis can be used throughout the testing cycle and is equally applicable at all testing phases. After determining the necessary test cases with equivalence partitioning and subsequent

boundary value analysis, it is necessary to define the combinations of the test cases when there are multiple inputs to a software component.

2.1.10 Branch/Condition Coverage Testing

Condition coverage combines the requirements for decision coverage with those for condition coverage. i.e. there must be sufficient test cases to toggle the decision outcome between true and false and to toggle each condition value between true and false [4]. Hence, a minimum of two test cases are necessary for each decision. Using the example (A or B), test cases (TT) and (FF) would meet the coverage requirement. However, these two tests do not distinguish the correct expression (A or B) from the expression A or from the expression B or from the expression (A and B).

2.1.11 Cause-Effect Graphing

In software testing, a cause-effect graph is a directed graph that maps a set of causes to a set of effects. The causes may be thought of as the input to the program, and the effects may be thought of as the output. Usually the graph shows the nodes representing the causes on the left side and the nodes representing the effects on the right side [19]. There may be intermediate nodes in between that combine inputs using logical operators such as AND & OR.

2.1.12 Comparison Testing

Comparison Testing means comparing your software with the better one or your Competitor. While comparison Testing we basically compare the Performance of the software. For ex If you have to do Comparison Testing of PDF converter (Desktop Based Application) then you will compare your software with your Competitor on the basis of [1]:

1. Speed of Conversion PDF file into Word.
2. Quality of converted file.

Similarly, we can compare various types of software with competitor’s software.

2.1.13 Compatibility Testing

Part of software non-functional tests, is testing conducted on the application to evaluate the application's compatibility with the computing environment. Computing environment may contain some or all of the below mentioned elements [1]:

- Computing capacity of Hardware Platform.
- Bandwidth handling capacity of networking hardware
- Compatibility of peripherals (Printer, DVD drive, etc.)
- Operating systems (UNIX, Windows, etc.)
- Database (Oracle, Sybase, DB2, etc.)
- Other System Software (Web server, networking/ messaging tool, etc.)
- Browser compatibility (Firefox, Netscape, Internet Explorer, Safari, etc.)

2.1.14 Control Testing

Control is a management tool to ensure that processing is performed in accordance to what management desire or intents of management. The objective of the control testing is accurate and complete data, authorized transactions, process meeting the needs of the user. Testers should have negative approach i.e. should determine or anticipate what can go wrong in the application system [19]. Develop risk matrix, which identifies the risks, controls; segment within application system in which control resides.

Example of control testing is file reconciliation procedures work and manual controls in place.

2.1.15 CRUD Testing

CRUD testing is actually Black Box testing. CRUD stands for (Create, Read, Update, and Delete). i.e., whether you can create or add data, whether you can read or access the data after it is saved once, or whether you can detect the data along with its relationship. Crude testing is nothing but exploratory testing and this is performed by the testers itself. Building a CRUD matrix and testing all object creations, reads, updates, and deletions [11].

2.1.16 Database Testing

Here database testing means test engineer should test the data integrity data accessing query retrieving modifications updating and deletion etc database can be tested in various ways, if we are using SQL server then open the SQL query analyzer and write the queries to retrieve the data. Then verify whether the expected result is correct or not. IF not the data is not inserted into database. We can play with queries to insert, update and delete the data from the data base and check in the front end of the

application [1]. If it is Oracle then open the SQL plus and follow the same procedure. Most of the time we find invalid data. Same way we can test the stored procedure in SQL query analyzer and check the result. We can play with queries to insert, update and delete the data from the data base and check in the front end of the application.

2.1.17 Decision Tables Testing

Decision tables are used to record complex business rules that must be implemented in the program, and therefore tested. A sample decision table is found in Table. In the table, the conditions represent possible input conditions. The actions are the events that should trigger, depending upon the makeup of the input conditions. Each column in the table is a unique combination of input conditions (and is called a rule) that result in triggering the action(s) associated with the rule [4]. Each rule (or column) should become a test case. If a player (A) lands on property owned by another player (B), A must pay rent to B. If A does not have enough money to pay B, A is out of the game.

Table 2.2: Decision Table

	Rule 1	**Rule 2**	**Rule 3**
Conditions			
A lands on B's property	Yes	Yes	No
A has enough money to pay rent	Yes	Yes	--
Actions			
A stays in game	Yes	No	Yes

2.1.18 Desk Checking

Similar to system testing the "macro" end of the test scale involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate. End to end testing is also called system testing. Done by senior test engineers or test lead end to end testing is nothing but inter system testing. If our application should share the data with other similar

applications, testing whether it is coordinating properly with other applications is called end to end testing [24].

2.1.19 Dynamic Testing

Dynamic testing is a term used in software engineering to describe the testing of the dynamic behaviour of code. That is dynamic analysis refers to the examination of the physical response from the system to variables that are not constant and change with time. In dynamic testing the software must actually be compiled and run; this is in contrast to static testing. Dynamic testing is the validation portion of Verification and Validation [1].

2.1.20 End-to-End Testing

End-to-end testing is the process of testing transactions or business level products as they pass right through the computer systems. Thus this generally ensures that all aspects of the business are supported by the systems under test. End to End Testing: is nothing but penetration testing. Whether our application build is co existing with other existing system or not is called end to end testing [1].

2.1.21 Error Handling Testing

Error handling refers to the anticipation, detection, and resolution of programming, application, and communications errors. Specialized programs, called error handlers, are available for some applications. The best programs of this type forestall errors if possible, recover from them when they occur without terminating the application, or (if all else fails) gracefully terminate an affected application and save the error information to a log file. In programming, a development error is one that can be prevented. Such an error can occur in syntax or logic. Syntax errors, which are typographical mistakes or improper use of special characters, are handled by rigorous proof reading. Logical errors, also called bugs, occur when executed code does not produce the expected or desired result. Logic errors are best handled by meticulous program debugging. This can be an ongoing process that involves, in addition to the traditional debugging routine, beta testing prior to official release and customer feedback after official release [5].

2.1.22 Exception Testing

Functional testing is defined as developing transactions to test every different combination of GOOD data. This ensures that all types/classes of good data will be properly processed by the system. Exception testing is defined as a complete and honest effort to break the system. With exception testing, the team tries to dream up every conceivable way that users will run bad data through the system and make sure that [4]:

1. The system does not crash.
2. The system is able to stop the bad data.
3. The system reports the root cause of the problem in an understandable way and suggests possible ways of fixing it.

2.1.23 Exploratory Testing

Exploratory testing is a method of manual testing that is concisely described as simultaneous learning, test design and test execution. Cem Kaner, who coined the term in 1983, now defines exploratory testing as "a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his/her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project". While the software is being tested, the tester learns things that together with experience and creativity generate new good tests to run [8]. Exploratory testing is often thought of as a black box testing technique. Instead, those who have studied it consider it a test approach that can be applied to any test technique, at any stage in the development process. The key is neither the test technique nor the item being tested or reviewed; the key is the cognitive engagement of the tester, and the tester's responsibility for managing his or her time. Exploratory testing is also known as ad hoc testing. Unfortunately, ad hoc is too often synonymous with sloppy and careless work.

2.1.24 Globalization Testing

The goal of globalization testing is to detect potential problems in application design that could inhibit globalization. It makes sure that the code can handle all international support without breaking functionality that would cause either data loss or display problems. Globalization testing checks proper functionality of the product with any of

the culture/locale settings using every type of international input possible. Proper functionality of the product assumes both a stable component that works according to design specification, regardless of international environment settings or cultures/locales, and the correct representation of data [4].

2.1.25 Grey Box Testing

Grey box testing is a software testing technique that uses a combination of black box testing and white box testing. Gray box testing is not black box testing, because the tester does know some of the internal workings of the software under test. In gray box testing, the tester applies a limited number of test cases to the internal workings of the software under test. In the remaining part of the gray box testing, one takes a black box approach in applying inputs to the software under test and observing the outputs. Gray box testing is a powerful idea [1]. The concept is simple; if one knows something about how the product works on the inside, one can test it better, even from the outside. Gray box testing is not to be confused with white box testing; i.e. a testing approach that attempts to cover the internals of the product in detail. Gray box testing is a test strategy based partly on internals. The testing approach is known as gray box testing, when one does have some knowledge, but not the full knowledge of the internals of the product one is testing. In gray box testing, just as in black box testing, you test from the outside of a product, just as you do with black box, but you make better-informed testing choices because you're better informed, because you know how the underlying software components operate and interact [8].

2.1.26 Incremental Integration Testing

Incremental integration testing is continuous testing of an application as new functionality is recommended. This may require that various aspects of an application's functionality are independent enough to work separately, before all parts of the program are completed, or that test drivers are developed as needed [22]. This type of testing may be performed by programmers, software engineers, or test engineers.

2.1.27 Inspections

An inspection is one of the most common sorts of review practices found in software projects. The goal of the inspection is for all of the inspectors to reach consensus on a

work product and approve it for use in the project. Commonly inspected work products include software requirements specifications and test plans. In an inspection, a work product is selected for review and a team is gathered for an inspection meeting to review the work product. A moderator is chosen to moderate the meeting. Each inspector prepares for the meeting by reading the work product and noting each defect. The goal of the inspection is to identify defects. In an inspection, a defect is any part of the work product that will keep an inspector from approving it. For example, if the team is inspecting a software requirements specification, each defect will be text in the document which an inspector disagrees with [19].

2.1.28 Integration Testing

While software modules may function well by themselves when they are developed, getting them to work together efficiently and correctly is another matter. After they have been coded and tested individually, individual software components are combined to form a final software product. During this integration effort, tests are performed on various groupings of components to determine how well they work together. Incompatibilities, errors, and inadequacies are discovered and fixed. Eventually, all software modules are integrated and debugged so they function correctly as a whole. Integration testing is the activity of software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before system testing [1].

2.1.29 Internationalization Testing

World is flat. If you are reading this page, chances are that you are experiencing this as well. It is very difficult to survive in the current world if you are selling your product in only one country or geological region. Even if you are selling in all over the world, but your product is not available in the regional languages, you might not be in a comfortable situation. Products developed in one location are used all over the world with different languages and regional standards. This arises the need to test product in different languages and different regional standards [1]. Multilingual and localization testing can increase your products usability and acceptability worldwide.

2.1.30 Intersystem Testing

There is software, some built in the OS and many after market that can do in system "house cleaning" and testing. One example is PC Alert. It shows up as a small open eye in task bar and when we move mouse over it, it reads out my CPU temp. If I right click it and go to the main page, I can test the inter system of every function of my computer as well as all the external systems as well. An inter system test, for example, could be a memory map, showing what is in memory at that moment. Another is a test of my fans, speed, efficiency etc. It basically is the way you find out if the inter system is working without any problem and is a big help when you do have a problem in finding a fix for it [1].

2.1.31 Load Testing

Load testing is the process of putting demand on a system or device and measuring its response. In mechanical systems it refers to the testing of a system to certify it under the appropriate regulations. Load testing is usually carried out to a load 1.5 times the Safe Working Load (SWL). Periodic recertification is required. The term load testing is used in different ways in the professional software testing community. Load testing generally refers to the practice of modeling the expected usage of a software program by simulating multiple users accessing the program concurrently. As such, this testing is most relevant for multi-user systems, often one built using a client/server model, such as web servers. However, other types of software systems can also be load tested. For example, a word processor or graphics editor can be forced to read an extremely large document; or a financial package can be forced to generate a report based on several years' worth of data [4]. The most accurate load testing occurs with actual, rather than theoretical, results. Load and performance testing is to test software intended for a multi-user audience for the desired performance by subjecting it with an equal amount of virtual users and then monitoring the performance under the specified load, usually in a test environment identical to the production environment, before going live [1].

2.1.32 Localization Testing

Localization is the process of customizing a software application that was originally designed for a domestic market so that it can be released in foreign markets. This process involves translating all native language strings to the target language and

customizing the GUI so that it is appropriate for the target market. Depending on the size and complexity of the software, localization can range from a simple process involving a small team of translators, linguists, engineers to a complex process requiring a Localization Project Manager directing a team of a hundred specialists. Localization is usually done using some combination of in-house resources, and full-scope services of a localization company [8].

2.1.33 Mutation Testing

Mutation testing is a method of software testing, which involves modifying program's source code in small ways. These, so-called mutations, are based on well-defined mutation operators that either mimic typical programming errors or force the creation of valuable tests. The purpose is to help the tester develop effective tests or locate weaknesses in the test data used for the program or in sections of the code that are seldom or never accessed during execution. Tests can be created to verify the correctness of the implementation of a given software system. But the creation of tests still poses the question whether the tests are correct and sufficiently cover the requirements that have originated the implementation. In this context, mutation testing was pioneered in the 1970s to locate and expose weaknesses in test suites [1]. The theory was that if a mutation was introduced without the behaviour (generally output) of the program being affected, this indicated either that the code that had been mutated was never executed (redundant code) or that the testing suite was unable to locate the injected fault. In order for this to function at any scale, a large number of mutations had to be introduced into a large program, leading to the compilation and execution of an extremely large number of copies of the program. This problem of the expense of mutation testing has reduced its practical use as a method of software testing [4].

2.1.34 Negative Testing

Testing the system using negative data is called negative testing. Negative testing is nothing but testing the build by giving the invalid input data [1] which yields all negative results this testing is done with the intent that of test should be failed. For example testing the password where it should be minimum of 8 characters so testing it using 6 characters is negative testing.

2.1.35 Orthogonal Array Testing

Orthogonal array testing is a systematic, statistical way of testing. Orthogonal arrays could be applied in user interface testing, system testing, regression testing, configuration testing and performance testing. All orthogonal vectors exhibit orthogonality. Orthogonal vectors exhibit the following properties [1]:

- Each of the vectors conveys information different from any other vector in the sequence, i.e., each vector conveys unique information therefore avoiding redundancy.
- On a linear addition, the signals may be separated easily.
- Each of the vectors is statistically independent from each other.
- When linearly added, the resultant is the arithmetic sum of the individual components.

2.1.36 Parallel Testing

As a test engineer, you may have explored ways to enhance test system performance in the past though parallel testing. However, the latest off-the-shelf test management software tools simplify parallel test system implementation. These tools increase test throughput and drive down test system costs. In general, parallel testing involves testing multiple products or subcomponents simultaneously. A parallel test station typically shares a set of test equipment across multiple test sockets, but, in some cases, it may have a separate set of hardware for each unit under test (UUT). The majority of nonparallel test systems test only one product or subcomponent at a time, leaving expensive test hardware idle more than 50 percent of the test time. Thus, with parallel testing, you can increase the throughput of manufacturing test systems without spending a lot of money to duplicate and fan out additional test systems [1].

2.1.37 Performance Testing

System performance is generally assessed in terms of response time and throughput rates under differing processing and configuration conditions. To attack the performance problems, there are several questions should be asked first how much application logic should be remotely executed, how much updating should be done to the database server over the network from the client workstation and how much data should be sent to each in each transaction. Performance testing is the process of

determining the speed or effectiveness of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time. Qualitative attributes such as reliability, scalability and interoperability may also be evaluated. Performance testing is often done in conjunction with stress testing. Performance testing can verify that a system meets the specifications claimed by its manufacturer or vendor. The process can compare two or more devices or programs in terms of parameters such as speed, data transfer rate, bandwidth, throughput, efficiency or reliability [1]. Effective performance testing can quickly identify the nature or location of a software-related performance problem.

2.1.38 Positive Testing

Testing the system using positive data is called positive testing. Positive testing is nothing but testing the build by giving the valid input data which yields all passed results. This testing is done with the intention of test should be passed [1]. For example testing the password where it should be minimum of 8 characters so testing it using 8 characters only is called positive testing.

2.1.39 Prototyping and Testing

A prototype is an initial version of a multimedia product. The prototype is tested to make sure it is fit for the audience and purpose. If there are any errors or problems, the prototype is improved and tested again. This goes on until the product is considered to be fully functional and suitable. This process is known as prototyping and testing and is illustrated in the flow diagram below [19]. A flow diagram showing the process of prototyping

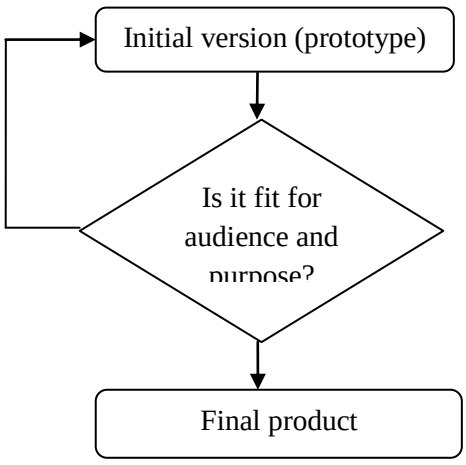


Fig 2.2: Process of prototyping

2.1.40 Random Testing

Random testing is a black-box technique and hence is useful when no information on the internal structure of the software can be used, so that the random values could be determined manually as well as by use of a pseudo-random number generator. To ensure the tests are repeatable, it is not acceptable to use a pseudo-random number generator which cannot be re-run to produce the same values. In practice, the use of the technique is most effective when the output from the result of each test can be automatically checked. In this situation, many tests can be run without manual intervention. In ideal circumstances, it may be possible to derive some reliability data from the result of random testing if it can be shown that the distribution used corresponds to that which would arise in actual use of the component [6].

2.1.41 Recovery Testing

In software testing, recovery testing is the activity of testing how well an application is able to recover from crashes, hardware failures and other similar problems. Recovery testing is the forced failure of the software in a variety of ways to verify that recovery is properly performed. Recovery testing should not be confused with reliability testing, which tries to discover the specific point at which failure occurs [15].

Examples of recovery testing: While an application is receiving data from a network, unplug the connecting cable. After some time, plug the cable back in and analyze the application's ability to continue receiving data from the point at which the network connection disappeared.

2.1.42 Regression Testing

Regression testing is any type of software testing which seeks to uncover regression bugs. Regression bugs occur whenever software functionality that previously worked as desired, stops working or no longer works in the same way that was previously planned. Typically regression bugs occur as an unintended consequence of program changes. Common methods of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged. A run-time error takes place during the execution of a program, and usually happens because of adverse system parameters or invalid input data [17]. An example is the lack of sufficient memory to run an application or a memory conflict with another program.

On the Internet, run-time errors can result from electrical noise, various forms of malware or an exceptionally heavy demand on a server. Run-time errors can be resolved, or their impact minimized, by the use of error handler programs, by vigilance on the part of network and server administrators, and by reasonable security countermeasures on the part of Internet users [1]

2.1.43 Requirements Testing

Requirements seem to be ephemeral. They flit in and out of projects; they are capricious, intractable, unpredictable and sometimes invisible. When gathering requirements we are searching for all of the criteria for a system's success. We throw out a net and try to capture all these criteria. Using Blitzing, Rapid Application Development (RAD), Joint Application Development (JAD), Quality Function Deployment (QFD), interviewing, apprenticing, data analysis and many other techniques, we try to snare all of the requirements in our net [1].

2.1.44 Risk-Based Testing

Testing is the means used in software development to reduce risks associated with a system. Risk-based testing (RBT) is a type of software testing that prioritizes the features and functions to be tested based on priority/importance and likelihood or impact of failure. In theory, since there is an infinite number of a possible test, any set of tests must be a subset of all possible tests. Test techniques such as boundary value analysis and state transition testing aim to find the area's most likely to be defective. So by using test techniques, a software test engineer is already selecting tests based on risk [18].

2.1.45 Sandwich Testing

Sandwich testing is a incremental testing approach of low level testing. It is combination of top-down (stubs) and bottom-up (drivers) approach and finally done integration testing of it. In another way it is a combination of top down and bottom up testing. It is a black box testing. It is a type of testing in which a test engineer can test the application without having the knowledge of functional part of it. Sandwich testing is also known as Hybrid approach in Integration testing [1].

2.1.46 Statement Coverage

Software developers and testers commonly use statement coverage because of its simplicity and availability in object code instrumentation technology. Of all the structural coverage criteria, statement coverage is the weakest, indicating the fewest number of test cases. Bugs can easily occur in the cases that statement coverage cannot see. The most significant shortcoming of statement coverage is that it fails to measure whether you test simple if statements with a false decision outcome. Experts generally recommend to only use statement coverage if nothing else is available [4].

2.1.47 State Transition Testing

State transition testing uses the same principles as the State Transition Diagramming design technique. State transition testing focuses on the testing of transitions from one state (e.g., open, closed) of an object (e.g., an account) to another state. Incorporate the state transition testing preparation into the test scenario building process [4]. For example, include tests of an account transitioning from open to closed and an account transitioning from closed to open (e.g., account closed in error). State transition is a black box testing technique. It is used to test the objects state and behaviour i.e. state transition testing can be defined as object = state + behaviour. Behaviour is the sequence of messages (or events) that an object accepts. State is a condition in which a system is waiting for one or multiple events transition represents change from one state to another caused by an event. An event is input that may cause a transition. An action is operation initiated because of a state change (occur on transitions) e.g. an ATM machine after entering the valid pin the user can move to any transaction like balance enquiry, withdrawal, pin change etc [6].

2.1.48 Static Testing

Static testing is a form of software testing where the software isn't actually used. This is in contrast to dynamic testing. It is generally not detailed testing but checks mainly for the sanity of the code, algorithm, or document. It is primarily syntax checking of the code or and manually reading of the code or document to find errors. This type of testing can be used by the developer who wrote the code, in isolation [1]. Code reviews, inspections and walkthroughs are also used. From the black box testing point of view, static testing involves review of requirements or specifications. Bugs

discovered at this stage of development are less expensive to fix than later in the development cycle.

2.1.49 Stress Testing

Stress testing deals with the quality of the application in the environment. The idea is to create an environment more demanding of the application than the application would experience under normal workloads. This is the hardest and most complex category of testing to accomplish and it requires a joint effort from all teams. A test environment is established with many testing stations. At each station, a script is exercising the system. These scripts are usually based on the regression suite. More and more stations are added, all simultaneous hammering on the system, until the system breaks. The system is repaired and the stress test is repeated until a level of stress is reached that is higher than expected to be present at a customer site. Race conditions and memory leaks are often found under stress testing. A race condition is a conflict between at least two tests. Each test works correctly when done in isolation. When the two tests are run in parallel, one or both of the tests fail. This is usually due to an incorrectly managed lock. A memory leak happens when a test leaves allocated memory behind and does not correctly return the memory to the memory allocation scheme. The test seems to run correctly, but after being exercised several times, available memory is reduced until the system fails [1].

2.1.50 Syntax Testing

It data-driven technique to test combinations of input syntax It uses the syntax of the component inputs as the basis for the design of the test case. Syntax testing is a static, black box testing technique for protocol implementations. Beizer proposes that one specify the syntax for the Protocol in a convenient notation such as BNF (Backnus-Naur Form). Mutations are then made to the syntactic elements, and the modified grammar is used to produce aberrant test vectors. As syntax and semantic errors are caught by developers during compilation or debugging, so syntax Testing can be White box testing [4].

2.1.51 System Testing

When the software, hardware, and other subsystems are complete, they in turn are integrated and tested as a system. This is the final development testing. Any problems

or errors discovered during systems testing are analyzed to determine which subsystems are at fault, then those subsystems are sent back for debugging, with its attendant code, unit, and integration testing [5]. The various levels of testing associated with development and how problems and errors feed back to earlier developmental stages. System testing evaluates the functionality of the system, including capabilities, compatibility, stability, performance, security, and reliability.

2.1.52 Thread Testing

Thread testing is the testing of system by running multiple users as one process. It is basically used for load testing. In simple terms, when testers focus on testing individual logical execution paths in context of entire system, it is called as Thread Testing. It is also an integration testing methodology that is gaining popularity. Infact there is an entire Thread-Based Approach to Testing which is used by large projects which are really time-constrained. Basically the application is broken down to Logical executable threads and each thread's ownership is assigned to a Developer - Tester combination. Each Thread has a status associated with it for a particular build. The thread enters the [1] "TESTABLE" state once developer releases to an integration build. We can maintain a shared DB to keep track of all the threads that have been conjured as a result of breaking down the application. At further stages, integrating individual threads is carried out which would be called as Integrated Thread Testing. Thread testing is a software technique that demonstrates key functional capabilities by testing a string of program units that accomplishes a specific business function in the application. A thread is basically a business transaction consisting of a set of functions. It is a single discrete process which threads throughout the whole system. The business transaction thread is then tested. Threads are in turn integrated and incrementally tested as subsystems and then the whole system is tested. This approach facilitates early systems and acceptance testing. Example Online fund Transaction.

2.1.53 Top-Down Testing

A top-down approach is essentially breaking down a system to gain insight into its compositional sub-systems. In a top-down approach an overview of the system is first formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in yet greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. A top-down model is often

specified with the assistance of "black boxes" that make it easier to manipulate. However, black boxes may fail to elucidate elementary mechanisms or be detailed enough to realistically validate the model. A bottom-up approach is piecing together systems to give rise to grander systems, thus making the original systems sub-systems of the emergent system. In a bottom-up approach the individual base elements of the system are first specified in great detail [24]. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed. “Organic strategies” may result in a tangle of elements and subsystems, developed in isolation and subject to local optimization as opposed to meeting a global purpose [1].

2.1.54 Unit Testing

In computer programming, unit testing is a procedure used to validate that individual units of source code are working properly. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method; which may belong to a base/super class, abstract class or derived/child class. Ideally, each test case is independent from the others; mock objects and test harnesses can be used to assist testing a module in isolation [24]. Unit testing is typically done by developers and not by Software testers or end-users.

2.1.55 Volume Testing

Volume Testing belongs to the group of non-functional tests, which are often misunderstood and/or used interchangeably. Volume testing refers to testing a software application for a certain data volume. This volume can in generic terms be the database size or it could also be the size of an interface file that is the subject of volume testing. For example, if you want to volume test your application with a specific database size, you will explode your database to that size and then test the application's performance on it. Another example could be when there is a requirement for your application to interact with an interface file this interaction could be reading and/or writing on to/from the file. You will create a sample file of the size you want and then test the application's functionality with that file to check performance [1].

2.1.56 White Box Testing

White box testing is performed based on the knowledge of how the system is implemented. White box testing includes analyzing data flow, control flow, information flow, coding practices, and exception and error handling within the system, to test the intended and unintended software behaviour [11].

White box testing requires access to the source code. Though white box testing can be performed any time in the life cycle after the code is developed, it is a good practice to perform white box testing during the unit testing phase. White box testing requires knowing what makes software secure or insecure, how to think like an attacker, and how to use different testing tools and techniques. The first step in white box testing is to comprehend and analyze source code, so knowing what makes software secure is a fundamental requirement. Second, to create tests that exploit software, a tester must think like an attacker. Third, to perform testing effectively, testers need to know the different tools and techniques available for white box testing. The three requirements do not work in isolation, but together [1].

Classification of Testing Types: Each testing technique has a unique purpose and the method in which it is applied. There are various type of testing technique which are discussed above are categorised in White Box, Black Box and Grey Box testing i.e. shows in table 2.3.

Table 2.3: Classification of testing technique

Software Testing Technique		
Black box testing	**White Box Testing**	**Grey Box**
Functional and system testing, Stress testing, Performance testing, Usability testing, Acceptance testing, Beta testing, Ad hoc Testing, Regression Testing, Intersystem Testing, Volume Testing, Parallel Testing, Boundary value testing, CRUD testing, End-to-end testing, Sandwich testing, State transition testing, syntax testing	Unit Testing, Error Handling Testing, Desk checking, Code walk-through, Code reviews and inspection, Code Coverage Testing, Statement/Path/Function/condition Testing, Complexity Testing / Cyclomatic complexity, Mutation Testing	Integration testing, Regression testing

Chapter 3: SDLC & SOFTWARE QUALITY

3.1 SDLC

SDLS refers to software development life cycle, i.e. the various stages used in the life cycle of software development. There are various software development approaches defined and designed which are used during development process of software, these approaches are also referred as "Software Development Process Models". In this chapter, various commonly known software development process models are discussed.

3.1.1 Waterfall Model

Waterfall approach was first process model to be introduced and followed widely in software engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate process phases. The phases in Waterfall model are: Requirement Specifications phase, Software Design, Implementation, Testing, Deployment of system & Maintenance [1]. All these phases are cascaded to each other so that second phase is started as and when defined set of goals are achieved for first phase and it is signed off, so the name "Waterfall Model". Fig 3.1 shows all the phases in water fall model.

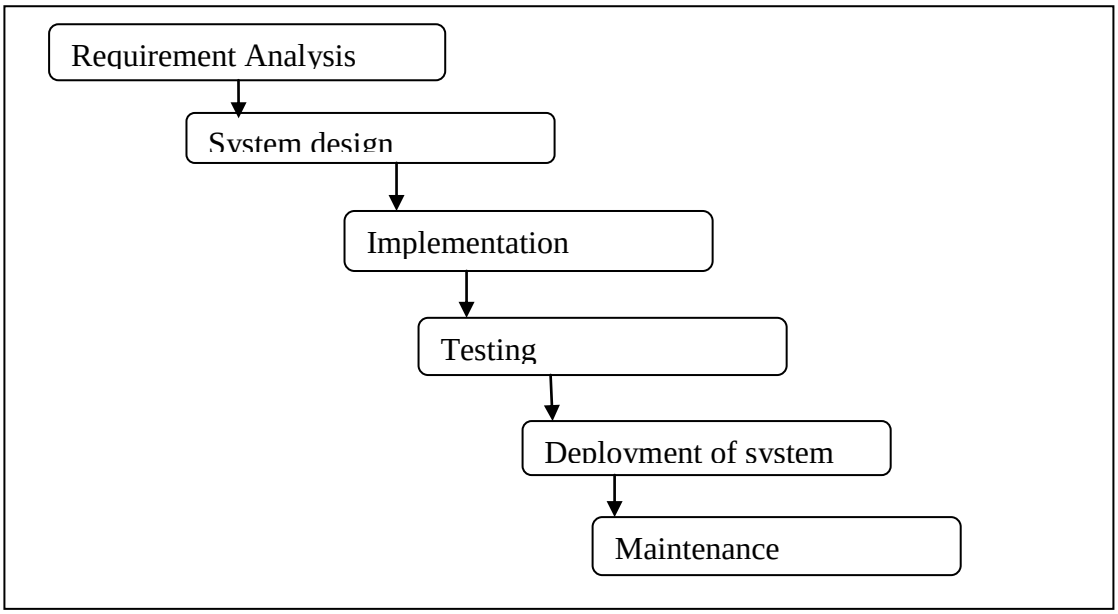


Fig 3.1: The phases of Waterfall Model

Advantages

In Waterfall model, every phase is implemented in a sequential order. Waterfall model is used where the duration of project is very less, and it is best suited for small projects. Also, Waterfall model is suitable when the specification and requirements are clearly stated for the software project.

Disadvantages

In Waterfall model, the output of one phase forms the input of the next phase. This concept actually turns as its disadvantage i.e. when a mistake occurs in a particular phase, the same mistakes gets carried over to the last phase. Waterfall model is time intensive process and almost provides little or no option to change user requirements. This model is useful only when the requirements are freezed.

3.1.2 Prototyping Model

A prototype is a working model that is functionally equivalent to a component of the product. In many instances the client only has a general view of what is expected from the software product [27]. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs and the output requirements, the prototyping model may be employed.

Advantages

With reduced time and costs, Prototyping can improve the quality of requirements and specifications provided to developers. Prototyping requires user involvement and allows them to see and interact with a prototype allowing them to provide better and more complete feedback and specifications.

Disadvantages

Insufficient analysis: The focus on a limited prototype can distract developers from properly analyzing the complete project. User can begin to think that a prototype, intended to be thrown away, it is actually a final system that merely needs to be finished or polished developer attachment to prototype.

3.1.3 Spiral Model

The spiral model, also known as the spiral lifecycle model, is a systems development method (SDM) used in information technology (IT) [1]. This model of development combines the features of the prototyping model and the waterfall model. The spiral model is intended for large, expensive, and complicated projects.

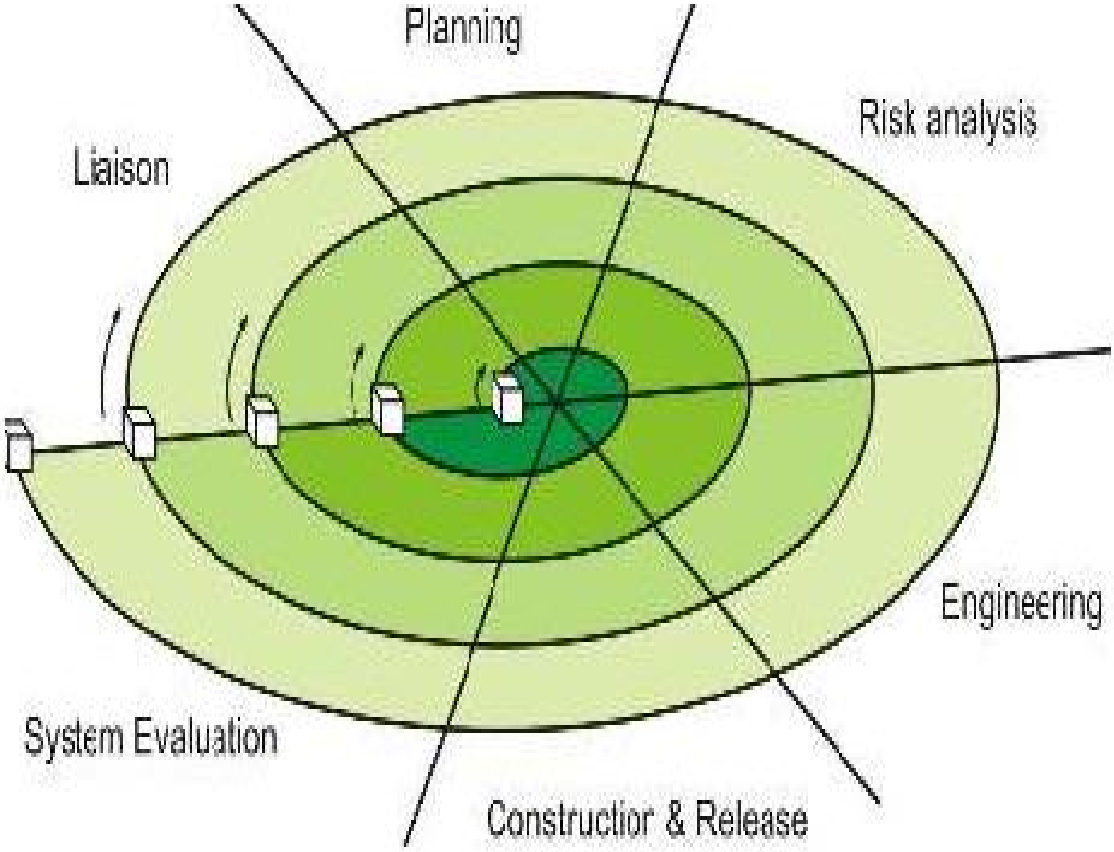


Fig 3.2: Phases of spiral model [28]

Advantages

Estimates (i.e. budget, schedule, etc.) become more realistic as work progresses, because important issues are discovered earlier. It is more able to cope with the (nearly inevitable) changes that software development generally entails. Software engineers (who can get restless with protracted design processes) can get their hands in and start working on a project earlier.

Disadvantages

As it is highly customized so there is limiting re-usability. It is applied differently for each application. There is a risk of not meeting budget or schedule.

3.1.4 Iterative Model

An iterative lifecycle model does not attempt to start with a full specification of requirements, which is the common scenario in all the software projects. Instead, development begins by specifying and implementing just a part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model [29]. Consider an iterative lifecycle model which consists of repeating the following four phases in sequence.

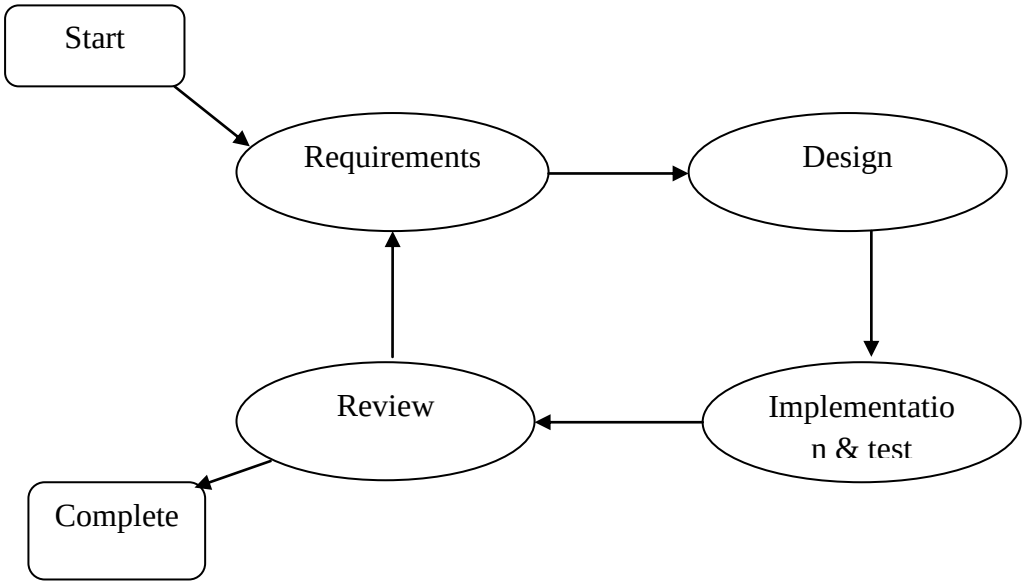


Fig 3.3: Iterative lifecycle

Advantages

Iterative enhancement model provides the benefits of producing software in multiple versions that allows better testing and quality control from lower to higher levels, hence providing better reliability.

Disadvantages

Each phase of an iterative model is rigid with no overlaps and costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle

3.1.5 RAD Model

RAD is a linear sequential software development process model that emphasis an extremely short development cycle using a component based construction approach. If the requirements are well understood and defines, and the project scope is constraint, the RAD process enables a development team to create a fully functional system with in very short time period [1].

Advantages

RAD reduces the development time and reusability of components help to speed up development. All functions are modularized so it is easy to work with.

Disadvantages

For large projects RAD require highly skilled engineers in the team. Both end customer and developer should be committed to complete the system in a much abbreviated time frame. If commitment is lacking RAD will fail. RAD is based on Object Oriented approach and if it is difficult to modularize the project the RAD may not work well.

3.1.6 V-Model

The V-model is a software development model which can be presumed to be the extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships [28] between each phase of the development lifecycle and its associated phase of testing.

Requirements analysis

In this phase, the requirements of the proposed system are collected by analyzing the needs of the user(s). This phase is concerned about establishing what the ideal system has to perform [1]. However, it does not determine how the software will be designed or built. Usually, the users are interviewed and a document called the user requirements document is generated. The users carefully review this document as this document would serve as the guideline for the system designers in the system design phase. The user acceptance tests are designed in this phase.

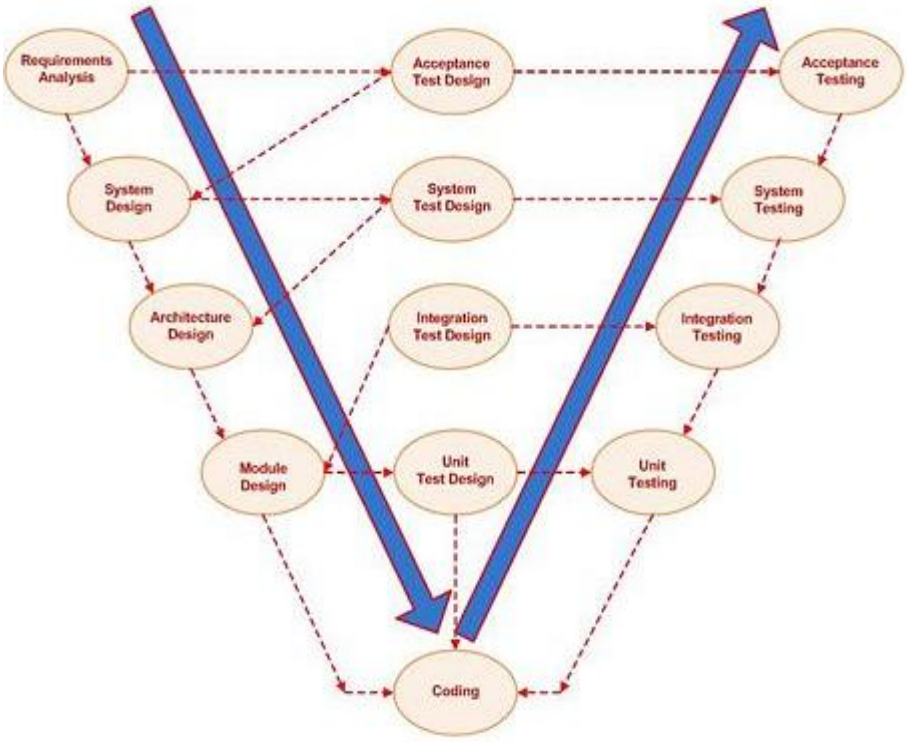


Fig 3.4: V-model [28]

System Design

System engineers analyze and understand the business of the proposed system by studying the user requirements document. They figure out possibilities and techniques by which the user requirements can be implemented. If any of the requirements are not feasible, the user is informed of the issue. The software specification document which serves as a blueprint for the development phase is generated. This document contains the general system organization, menu structures, data structures etc. It may also hold example business scenarios, sample windows, reports for the better understanding. The documents for system testing are prepared in this phase [1].

Architecture Design

This phase can also be called as high-level design. The baseline in selecting the architecture is that it should realize all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology details etc. The integration testing design is carried out in this phase [1].

Module Design

This phase can also be called as low-level design. The designed system is broken up in to smaller units or modules and each of them is explained so that the programmer can start coding directly. The low level design document or program specifications will contain a detailed functional logic of the module, in pseudo code - database tables, with all elements, including their type and size all interface details with complete API reference all dependency issues- error message listings complete input and outputs for a module. The unit test design is developed in this stage [1].

Advantages

Verification and validation performed simultaneously. We can save the time duration. This is because the Testing activities start as soon as the customer gives the requirement. Cost will be lesser, V model is cost saving because there is early detection of bugs while development and unit testing.

Disadvantages

Expensive, for big projects it's a repeated process the customer is involved after end product is finished. Any risk/contingencies are not analysed during the v-model.

3.2 Software Quality

Everyone is committed to quality; however, the following statement shows some of the confusing ideas shared by many individuals that inhibit achieving a quality commitment: Quality requires a commitment, particularly from top management. Close cooperation of management and staff is required in order to make it happen.

- Many individuals believe that defect-free products and services are impossible, and accept certain levels of defects as normal and acceptable [1].
- Quality is frequently associated with cost, meaning that high quality equals high cost. This is confusion between quality of design and quality of conformance [2].
- Quality demands requirement specifications in enough detail that the products produced can be quantitatively measured against those specifications. Many organizations are not capable or willing to expend the effort to produce specifications at the level of detail required [3].

- Technical personnel often believe that standards stifle their creativity, and thus do not abide by standards compliance. However, for quality to happen, well-defined standards and procedures must be followed [1].

Quality cannot be achieved by assessing an already completed product. The aim therefore, is to prevent quality defects or deficiencies in the first place, and to make the products assessable by quality assurance measures. Some quality assurance measures include: structuring the development process with a software development standard and supporting the development process with methods, techniques, and tools. The undetected bugs in the software that caused millions of losses to business have necessitated the growth of independent testing, which is performed by a company other than the developers of the system [8].

In addition to product assessments, process assessments are essential to a quality management program. Examples include documentation of coding standards, prescription and use of standards, methods, and tools, procedures for data backup, test methodology, change management, defect documentation, and reconciliation. Quality management decreases production costs because the sooner a defect is located and corrected, the less costly it will be in the long run [7]. With the advent of automated testing tools, although the initial investment can be substantial, the long-term result will be higher-quality products and reduced maintenance costs. The total cost of effective quality management is the sum of four component costs: prevention, inspection, internal failure, and external failure. Prevention costs consist of actions taken to prevent defects from occurring in the first place. Inspection costs consist of measuring, evaluating, and auditing products or services for conformance to standards and specifications [9]. Internal failure costs are those incurred in fixing defective products before they are delivered.

3.2.1 Quality Attributes

Quality can be measured using various quality attributes. Common ones are discussed here:

- **Understandability:** The purpose of the software product is clear. This goes further than just a statement of purpose all of the design and user documentation

must be clearly written so that it is easily understandable. Obviously, the user context must be taken into account, e.g. if the software product is to be used by software engineers it is not required to be understandable to lay users [1].

- **Completeness:** All parts of the software product are present and each of its parts are fully developed. For example, if the code calls a sub-routine from an external library, the software package must provide reference to that library and all required parameters must be passed. All required input data must be available [1].

- **Conciseness:** No excessive information is present. This is important where memory capacity is limited, and it is important to reduce lines of code to a minimum. It can be improved by replacing repeated functionality by one sub-routine or function which achieves that functionality. This quality factor also applies to documentation [26].

- **Portability:** The software product can be easily operated or made to operate on multiple computer configurations [28]. This can be between multiple hardware configurations (such as server hardware and personal computers), multiple operating systems (e.g. Microsoft Windows and Linux-based operating systems), or both.

- **Consistency:** The software contains uniform notation, symbology and terminology within itself [1].

- **Maintainability:** The product should facilitates updating to satisfy new requirements and software product that is maintainable is simple, well-documented [26].

- **Testability:** The software product facilitates the establishment of acceptance criteria and supports evaluation of its performance. Such a characteristic must be built-in during the design phase if the product is to be easily testable, since a complex design leads to poor testability [26].

- **Usability:** The product is convenient and practicable to use. The component of the software which has most impact on this is the user interface (UI), which for best usability is usually graphical [26].

- **Reliability:** The software can be expected to perform its intended functions satisfactorily over a period of time. Reliability also encompasses environmental considerations in that the product is required to perform correctly in whatever conditions it is operated in; this is sometimes termed robustness [26].

- **Structure:** The software possesses a definite pattern of organization in its constituent parts [1].

- **Efficiency:** The software product fulfils its purpose without wasting resources, e.g. memory or CPU cycles [26].

- **Security:** The product is able to protect data against unauthorized access and to withstand malicious interference with its operations. Besides the presence of appropriate security mechanisms such as authentication, access control and encryption, security also implies reliability in the face of malicious, intelligent and adaptive attackers [26].

In order to measure quality, we need to analyse requirements to design test cases, then design the test cases, document them, implement them and execute these test cases. Then the results are analysed. Before all this, we need to plan for testing, including risk analysis and test management practices. An example is IBM RUP software tools used by testers to execute a software test plan [11]. This all includes communication skill for the effective tester.

This chapter summarized various software development models and various quality attributes. Next chapter describes the statement of problem taken for this thesis work.

Chapter 4: Problem Statement

4.1 Problem Definition

The study of various software development process models reveal that in almost all these models, software testing is included as one phase, but testing is required at each phase and not at a particular stage. The main purpose of software testing is to uncover errors which are not simply syntax errors in code but various other types of errors in all the documents produced during the software development ,e.g. software requirements document, design document, test plan etc. Various types of software testing techniques have been developed till date, but which type of testing technique will be suitable and sufficient for checking a particular document in which phase of software development life cycle (SDLC) is not yet clear. So here the problem is to

1. Identify the testing techniques which can be applied at different levels and phases of software development life cycles

Also, software quality is an essential part of any software project. Various quality assurance and control activities may be used to ensure quality in the software project. Different quality attributes need different types of testing to measure software quality. The problem is that out of numerous testing techniques possible, which testing technique should be applied to measure which quality attribute is not very clear. So the next problem takes here is

2. Identify the testing techniques which can be applied to measure which software quality attribute

4.2 Justification

By categorizing which type of testing to be applied at which phase of software development will help us plan for testing in that phase efficiently and to take full advantage of all the types of testing techniques to improve quality in that phase and consequently the overall quality of the software project. The relation between various quality attributes and the testing techniques required for each of these will help save

time and produce quicker results and streamlined testing of the project for that particular software quality attribute.

This chapter define the statement of problem. Next chapter describes the Proposed Solutions according to the problem of statement.

Chapter 5: Proposed Solutions

According to the problem statement in this report, a model "Software Development Life Cycle Testing Model" is proposed in which all types of testing techniques related to test all phases of SDLC are specified. V model of testing given by Mr. Perry includes only 5 phases of SDLC. Here this model is extended to include more phases of SDLC and select the types of testing technique that can be applied in each phase.

5.1 Apply Testing on all Phases of SDLC

It has always been a big question when to start testing. Experts suggest that every step taken in the development of the system must be tested thoroughly in a formal manner. It means that testing must be done for requirements gathering, designing, coding, and even for testing phase. Testing of testing efforts may seem to be unusual and surprising but it is an important effort because one needs to be sure about the testing efforts to be able to rely on its reports. A good testing life cycle begins during the requirements elicitation phase of software development, and concludes when the product is ready to install or ship, following a successful system test. Fig 6.1 given below shows that testing applied on all the phases (Requirement gathering, Designing, Coding, Testing, Implementation and Maintenance) of SDLC, not a particular stage. The study of various software development process models reveal that in almost all these models, software testing is included as one phase, but testing is required at each phase and not at a particular stage. In this SDLC testing model we applied the testing at all the phases of SDLC. By categorizing which type of testing technique to be applied at which phase of software development life cycle will help us plan for testing in that phase efficiently and to take full advantage of all the types of testing techniques to improve quality in that phase and consequently the overall quality of the software project. Well-defined traceable and controllable processes are required for enhancing the quality of the software products and gaining optimum benefits from applied effort. Software process is a stepwise sequence of activities carried with the focus of producing quality software in an economic manner it will be possible when we applied testing at all the phases of software development life cycle.

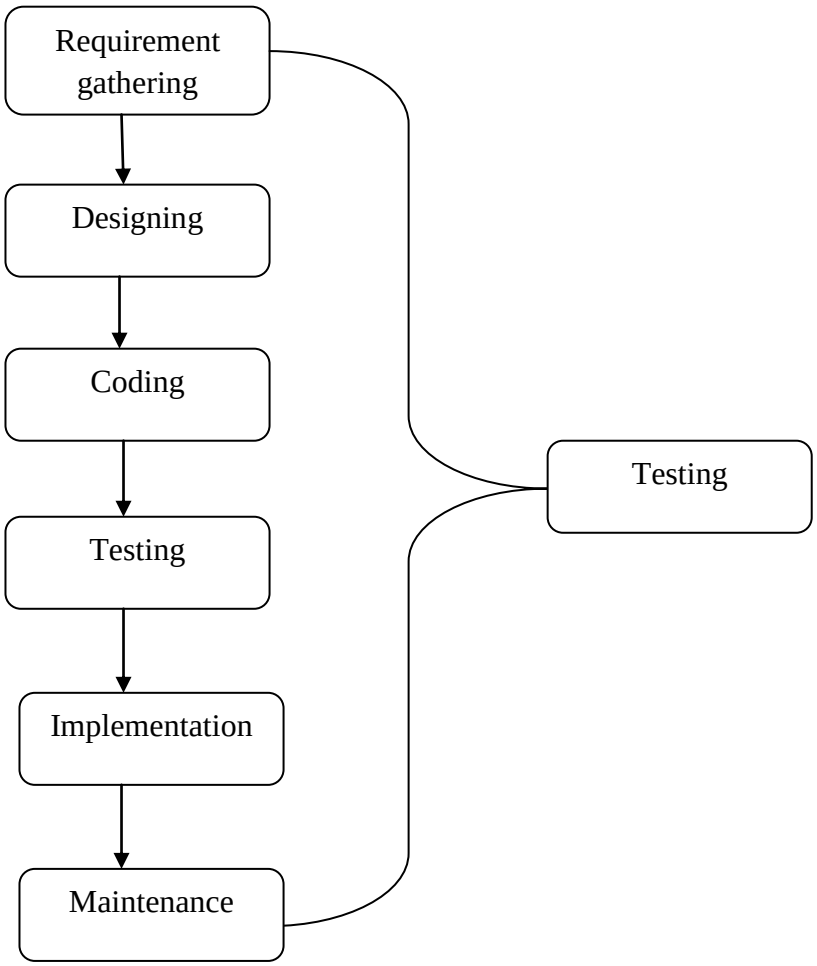


Fig 5.1: Applying testing on all phases of SDLC

Software testing is recommended to be started as early as possible in the earliest phases of the SDLC, most preferably in the requirement analysis phase itself and should be performed by skilled testers only and not by developers. Software development life cycle (SDLC) processes involve activities of software requirements analysis, requirements specification, design, coding, testing, delivery, and maintenance. The testing phase can be used in all of these life cycle phases as an umbrella activity.

5.2 Identifying Testing Techniques according to Phase of SDLC

We identifying that which type of testing technique can be applied to which phase of SDLC. Fig 5.2 shows the phases of SDLC and according to testing technique.

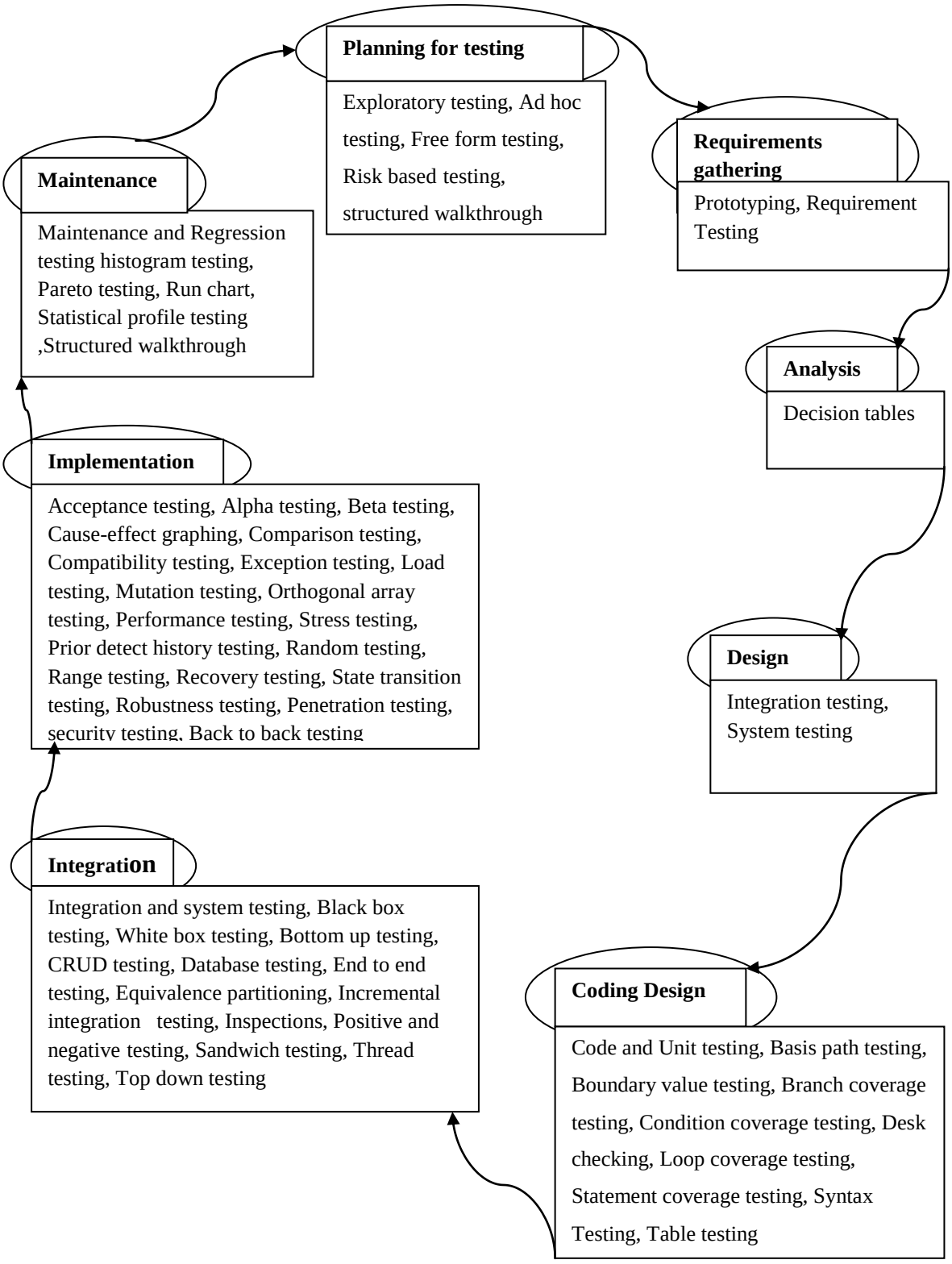


Fig 5.2: SDLC Testing Model

5.3 Application of Testing to Measurement of Quality Attributes

Different quality attributes need different types of testing to measure software quality. Various types of testing according to the quality feature it applies to in the table 6.3. In given table we identified that for a particular software quality feature which type of software testing technique can be applied:

Table 5.1: Testing Technique According to Quality Features

Quality Attribute	**Types of Testing**
Functionality	Functional testing
Security	Security testing
Complexity	Unit testing
Performance	Performance testing
Compatibility	Compatibility testing
Reliability	Stress testing, Robustness testing, load testing
Vulnerability	Penetration testing
Usability	Comparison testing
Consistency	Database testing, Table testing
Correctness	Database testing, Table testing
Portability	Portability Testing

Recovery	Recovery testing
Completeness	Boundary/Statement/Loop/Condition/ Path coverage testing
Efficiency	Performance testing
Understandability	Usability Testing
Structure	Structural testing
Maintainability	Regression testing

Chapter 6: CONCLUSION AND FUTURE SCOPE

Software testing is the activity that executes software with an intention of finding errors in it. Testing should be performed at different levels, including module level testing, unit level testing, interface testing and system level testing. Testing is done both at developer end and customer end and it is performed by testers as well as the customer before delivery of the product but it can ensure a fair level of confidence in the predictable behaviour of the product in the provided conditions.

Quality is the main focus of any software engineering project. Without measuring, we cannot be sure of the level of quality in software. So the methods of measuring the quality are software testing techniques. This thesis report relates various types of testing technique that we can apply in measuring various quality attributes. Also which testing are related to various phase of SDLC. General SDLC processes are applied to different type of projects under different conditions and requirements. There are various type of SDLC model (Waterfall Model, RAD Mode, Iterative Model, Proto Type Model, Spiral Model, V-Model, etc). But in all these models, testing is applied after a particular stage and not in all the phases. In this thesis report, it is concluded that testing should be applied in all the phases of SDLC and not at a particular stage. Which type of testing technique can be applied to which type of SDLC phase is also summarized.

Future work for this area will be to take more new coming testing techniques and relating these to the phases of SDLC. This will help taking the maximum advantage of that testing technique. And this will be helpful to conclude that.

REFERENCES

[1] Jain Deepak, "Software Engineering Principle and Practices" First edition by Oxford University Press, ISBN-13: 978-0-19-569484-0, (2009).

[2] Bersoff, E.H. and A.M. Davis, "Impact of Lifecycle Modes on Software Configuration Management", ACM, pp104-108 (1991).

[3] Boris Beizer, "Software testing techniques", Second edition, (1990).

[4] Chilarege, Ram, "Software Testing Best Practises" Center for Software Engineering, IBM Research (1999).

[5] Joe W. Duran, Semeon, C. Ntafos, "An Evaluation of Random Testing ", IEEE Transactions on Software engineering,Vol.SE-10,No.4, pp438-443 (July 1984).

[6] Beizer, Boris, "Black-Box Testing Technique for Functional Testing of Software and System" New York Wiley, ISBN: 0471120944 Physical description: xxv, 194 p. ill.; 23cm (1995).

[7] Barber, Scott "Software Testing: An Introduction", PerfTestPlus (2006).

[8] Cem Karner, "Testing Computer Software", (1993).

[9] IEEE "Standard Glossary of Software Engineering Terminology" (IEEE Std 610.12-1990), IEEE Computer society, (dec.10, 1990).

[10] Ballista COTS "Software Robustness Testing Harness" (1999).

[11] Kropp, N P Koopman, P J Siewiorek D P "Automated Robustness Testing of the-Shelf Software Component" 28th Annual International Symposium on Fault- Tolerant Computing (1995).

[12] Software testing slide of RST Corporation by Jeff Voas.

[13] Philip Koopman, John Devale. “Comparing the Robustness of POSIX Operation Systems”. Proceedings of FTCS’99, Madison, Wisconsin. June (1999).

[14] “Software Program Managers Network” Little Book of Testing, Vol. 1, (1998).

[15] Thompson, H. and J. Whittaker, “Testing for Software Security”, Dr. Dobbs Journal, (November 2002).

[16] Kit, Ed, “Software Testing in the Real World”, Addison-Wesley, 1 p.3 (1995).

[17] Jorgensen, Paul C., “Software Testing A Craftsman’s Approach”, CRC Press, p.3 (1995).

[18] “Software Program Managers Network”, Little Book of Testing, Vol. II, (1998).

[19] “Program Manager’s Guide for Managing Software”, 0.6, (29 June 2001).

[20] A. Bertolino, “Chapter 5: Software Testing” in IEEE SWEBOK Trial Version 1.00, (May 2001).

[21] IEEE, “IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology” (1990).

[22] IEEE, “ANSI/IEEE Standard 1008-1987, IEEE Standard for Software Unit Testing” (1986).

[23] Boehm, B.W., and others, “Characteristics of Software Quality,” TRW Software series, (December 22, 1973).

[24] IEEE “Standard for a Software Quality Metrics Methodology” (IEEE Standard P-1061/D21). New York, N.Y.: Institute of Electrical and Electronic Engineers, Inc, (1990).

[25] Pressman “Software Engineering: A Practitioner's Approach”. New York: McGraw-Hill, (1992).

[26] Pressman “Software Engineering: A Beginner's Guide”. New York McGraw-Hill, (Chapters 4, 5 & Appendix B) (1988).

[27] B A Wichmann, National Physical Laboratory, Teddington, Middlesex, TW11 0LW, UK (May 1998).

[28] Aggarwal K.K. and Yogesh Singh “Software Engineering”, New Age Publishers, New Delhi (2005).

[29] Pressman Roger S. “Software Engineering: A Practitioner’s Approach”, 6th ed., McGraw Hill (2005).

LIST OF PUBLICATIONS

1. Youddha Beer Singh, Shivani Goel "Role of Testing In Phases of SDLC and Quality" in International Journal of Information Technology & Knowledge Management. **[Accepted** for Vol.-II, Issue-II Dec. 2009**]**

2. Youddha Beer Singh, Shivani Goel "Role of Software Testing in Software Quality" at National conference on Modern Management Practices & Information technology Trends (MMPITT-09) held at Department of Business Management & Information Technology, DAV Institute of Engineering & Technology, Jalandhar(India) on April 17-18, 2009, PP 500-504.**[Presented & Published]**