T.Y.B.SC.(IT)

SEMESTER - VI (CBCS)

SOFTWARE QUALITY ASSURANCE

SUBJECT CODE : 88701

© UNIVERSITY OF MUMBAI

Prof. (Dr.) D. T. Shirke
Offg. Vice Chancellor
University of Mumbai, Mumbai.

Prin. Dr. Ajay Bhamare
Offg. Pro Vice-Chancellor,
University of Mumbai.

Prof. Prakash Mahanwar
Director
IDOL, University of Mumbai.

Programe Co-ordinator : **Shri. Mandar Bhanushe**
Head, Faculty of Science and Technology,
IDOL, University of Mumbai – 400098.

Course Co-ordinator : **Ms. Gouri Sawant**
Assistant Professor, B.Sc. IT, IDOL,
University of Mumbai - 400098.

Editor : **Dr. T.Parimalam**
Associate Professor and Head in Computer Science,
Nandha Arts and Science College,
Erode, 638052.

Course Writers : **Vinay Vikas Shahapurkar**
Assistant Professor,
Bunts Sangha's S.M. Shetty College of Science,
Commerce and Management Studies, Powai.

: **Avneet Kaur**
Assistant Professor,
Bunts Sangha's S.M. Shetty College of Science,
Commerce and Management Studies, Powai.

: **Dr. Triveni Koul**
Assistant Professor,
Yashwantrao Chavan College of Arts Commerce and Science,
Koper Khairane.

: **Sameera Salim Ibrahim**
Assistant Professor,
SIES (Nerul) College of Arts, Science & Commerce.

: **Akshata Laddha**
Assistant Professor,
Dilkap Research Institute of Engineering and
Management Studies, Neral.

October 2022, Print I

Published by
Director
Institute of Distance and Open Learning, University of Mumbai,Vidyanagari, Mumbai - 400 098.

DTP COMPOSED AND PRINTED BY
Mumbai University Press,
Vidyanagari, Santacruz (E), Mumbai - 400098.

CONTENTS

*****

B. Sc. (Information Technology)		Semester – VI	
Course Name: Software Quality Assurance		**Course Code: 88701**	
Periods per week (1 Period is 50 minutes)		**5**	
Credits		**2**	
		Hours	**Marks**
Evaluation System	**Theory Examination**	**2½**	**75**
	Internal	**--**	**25**

Unit	Details	Lectures
I	**Introduction to Quality:** Historical Perspective of Quality, What is Quality? (Is it a fact or perception?), Definitions of Quality, Core Components of Quality, Quality View, Financial Aspect of Quality, Customers, Suppliers and Processes, Total Quality Management (TQM), Quality Principles of Total Quality Management, Quality Management Through Statistical Process Control, Quality Management Through Cultural Changes, Continual (Continuous) Improvement Cycle, Quality in Different Areas, Benchmarking and Metrics, Problem Solving Techniques, Problem Solving Software Tools. **Software Quality:** Introduction, Constraints of Software Product Quality Assessment, Customer is a King, Quality and Productivity Relationship, Requirements of a Product, Organisation Culture, Characteristics of Software, Software Development Process, Types of Products, Schemes of Criticality Definitions, Problematic Areas of Software Development Life Cycle, Software Quality Management, Why Software Has Defects? Processes Related to Software Quality, Quality Management System Structure, Pillars of Quality Management System, Important Aspects of Quality Management.	**12**
II	**Fundamentals of testing:** Introduction, Necessity of testing, What is testing? Fundamental test process, The psychology of testing, Historical Perspective of Testing, Definitions of Testing, Approaches to Testing, Testing During Development Life Cycle, Requirement Traceability Matrix, Essentials of Software Testing, Workbench, Important Features of Testing Process, Misconceptions About Testing, Principles of Software Testing, Salient Features of Good Testing, Test Policy, Test Strategy or Test Approach, Test Planning, Testing Process and Number of Defects Found in Testing, Test Team Efficiency, Mutation Testing, Challenges in Testing, Test Team Approach, Process Problems Faced by Testing, Cost Aspect of Testing, Establishing Testing Policy, Methods, Structured Approach to Testing, Categories of Defect, Defect, Error, or Mistake in Software, Developing Test Strategy, Developing Testing Methodologies (Test Plan), Testing Process, Attitude Towards Testing (Common People Issues), Test Methodologies/Approaches, People Challenges in Software Testing, Raising Management Awareness for Testing, Skills Required by Tester,	**12**

	Testing throughout the software life cycle, Software development models, Test levels, Test types, the targets of testing, Maintenance testing	
III	**Unit Testing: Boundary Value Testing:** Normal Boundary Value Testing, Robust Boundary Value Testing, Worst-Case Boundary Value Testing, Special Value Testing, Examples, Random Testing, Guidelines for Boundary Value Testing, **Equivalence Class Testing:** Equivalence Classes, Traditional Equivalence Class Testing, Improved Equivalence Class Testing, Edge Testing, Guidelines and Observations. **Decision Table–Based Testing:** Decision Tables, Decision Table Techniques, Cause-and-Effect Graphing, Guidelines and Observations, **Path Testing:** Program Graphs, DD-Paths, Test Coverage Metrics, Basis Path Testing, Guidelines and Observations, **Data Flow Testing:** Define/Use Testing, Slice-Based Testing, Program Slicing Tools.	**12**
IV	**Software Verification and Validation:** Introduction, Verification, Verification Workbench, Methods of Verification, Types of reviews on the basis od Stage Phase, Entities involved in verification, Reviews in testing lifecycle, Coverage in Verification, Concerns of Verification, Validation, Validation Workbench, Levels of Validation, Coverage in Validation, Acceptance Testing, Management of Verification and Validation, Software development verification and validation activities. **V-test Model:** Introduction, V-model for software, testing during Proposal stage, Testing during requirement stage, Testing during test planning phase, Testing during design phase, Testing during coding, VV Model, Critical Roles and Responsibilities.	**12**
V	**Levels of Testing:** Introduction, Proposal Testing, Requirement Testing, Design Testing, Code Review, Unit Testing, Module Testing, Integration Testing, Big-Bang Testing, Sandwich Testing, Critical Path First, Sub System Testing, System Testing, Testing Stages. **Special Tests:** Introduction, GUI testing, Compatibility Testing, Security Testing, Performance Testing, Volume Testing, Stress Testing, Recovery Testing, Installation Testing, Requirement Testing, Regression Testing, Error Handling Testing, Manual Support Testing, Intersystem Testing, Control Testing, Smoke Testing, Adhoc Testing, Parallel Testing, Execution Testing, Operations Testing, Compliance Testing, Usability Testing, Decision Table Testing, Documentation Testing, Training testing, Rapid Testing, Control flow graph, Generating tests on the basis of Combinatorial Designs, State Graph, Risk Associated with New Technologies, Process maturity level of Technology, Testing Adequacy of Control in New technology usage, Object Oriented Application Testing, Testing of Internal Controls, COTS Testing, Client Server Testing, Web Application Testing, Mobile Application Testing, eBusiness eCommerce Testing, Agile Development Testing, Data Warehousing Testing.	**12**

Books and References:					
Sr. No.	**Title**	**Author/s**	**Publisher**	**Edition**	**Year**
1.	Software Testing and Continuous Quality Improvement	William E. Lewis	CRC Press	Third	2016
2	Software Testing: Principles, Techniques and Tools	M. G. Limaye	TMH		2017
3.	Foundations of Software Testing	Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black	Cengage Learning	3rd	
4.	Software Testing: A Craftsman's Approach	Paul C. Jorgenson	CRC Press	4th	2017

UNIT - I

1

INTRODUCTION TO QUALITY

Unit Structure

1.0 OBJECTIVES

Objectives of Quality Assurance are as follows:

- It helps monitor the software development method and the final software developed.
- It guarantees that the software project is implementing the standards and procedures set by the management.
- It helps in the notification of groups and individuals about its events and consequences.

- It guarantees that the problems, which are not solved within the software, are addressed by the higher administration.
- It helps recognize and fix shortages in the product, process, or standards.

1.1 INTRODUCTION

Evolution of mankind can be seen as a continuous effort to make things better and convenient through improvements linked with inventions of new products. Since time immemorial, humans have used various products to enhance their lifestyle. Initially, mankind was completely dependent on nature for satisfying their basic needs for food, shelter and clothing. With evolution, human needs increased from the basic level to substantial addition of derived needs to make life more comfortable. As civilisation progressed, humans started converting natural resources into things which could be used easily to satisfy their basic as well as derived needs.

Earlier, product quality was governed by the individual skill and it differed from instance to instance depending upon the creator and the process used to make it at that instance. Every product was considered as a separate project and every instance of the manufacturing process led to products of different quality attributes. Due to increase in demand for same or similar products which were expected to satisfy same/similar demands and mechanisation of the manufacturing processes, the concept of product specialisation and mass production came into existence. Technological developments made production faster and repetitive in nature.

Now, we are into the era of specialised mass production where producers have specialised domain knowledge and manufacturing skill required for particular product creation, and use this knowledge and skill to produce the product in huge qualities. The market is changing considerably from monopoly to fierce competition but still maintaining the individual product identity in terms of attributes and characteristics. There are large number of buyers as well as sellers in the market, providing and demanding similar products, which satisfy similar demands. Products may or may not be exactly the same but may be similar or satisfying similar needs/demands of the users. They may differ from one another to some extent on the basis of their cost, delivery schedule to acquire it, features, functionalities present/absent, etc. These may be termed as attributes of the quality of a product.

We will be using the word ‘product’ to represent ‘product’ as well as ‘service’, as both are intended to satisfy needs of users/customers. Service may be considered as a virtual product without physical existence but satisfying some needs of users.

1.2 HISTORICAL PERSPECTIVE OF QUALITY

Quality improvement is not a new pursuit for mankind. The field of quality and quality improvement has its roots in agriculture. Early efforts

of quality improvement in agriculture may be attributed to statistical research conducted in Britain, in early 20th century, to assist farmers in understanding how to plan the crops and rotate the plan of cultivation to maximise agricultural production while maintaining the soil quality at the same time.

This work inspired Walter Shewhart at Bell Laboratories to develop quality improvement programs through planned efforts. He adopted the concepts developed initially for agriculture to implement quality improvement programs for products and to reduce the cost to customer without affecting profitability for the manufacturer. Changes brought in by Walter Shewhart motivated Dr Edward Deming to implement quality improvement programs as a way to improve product quality. He devoted his life to teaching and improving quality methods and practices across the world through 'Total Quality Management' methodology.

Dr. Deming demonstrated his ideas of 'Total Quality Management' through continual improvement in Japan. Dr Joseph Juran also implemented quality improvement through measurement programs using different quality tools for assessment and improvement. Japanese producers fully embraced quality improvement methodologies and started to integrate the concepts of 'Total Quality Management' in their industries. The Dramatic improvement in quality of products in Japan after 1950 due to the revolutionary ideas of continual improvement through process measurement are still considered legendary.

During the last few decades, the Japanese industry has successfully utilised quality tools and 'Total Quality Management' methodologies as part of their successful effort to become a leading nation in a manufacturing and supplying a vast array of electronics, automotive and other products to the entire world. Quality of the products is established and continually improved in terms of features, consistent performance, lesser costs, reasonable delivery schedule, etc. in order to enhance the satisfaction of the customer. Japanese products started dictating the quality parameters in world market to the extent that many nations adopted quality improvement programs at national level to face the competition from the Japanese industry. Quality of Japanese products stems from the systematic organisation and understanding of processes used in all aspects of product development, and introduction of tools and methodologies that permit monitoring and understanding about what is happening in different processes of manufacturing and management of interactions of those processes. Japanese quality improvement programs created the sets of interrelated processes which assure the same product quality in repetitive manner and in large number to satisfy the demand of a huge market. Defects are analysed and root causes of the defects are identified and eliminated through continual process improvement. This has helped in optimising the processes to produce better results in repetitive manner.

1.3 WHAT IS QUALITY? (IS IT A FACT OR PERCEPTION?)

What is quality, is an important question but does not have a simple answer. Some people define it as a fact while others define it as a perception of customer/user.

We often talk about quality of a product to shortlist or select the best product among the equals when we wish to acquire one. We may not have a complete idea about the meaning of quality or what we are looking for while selecting a product, if somebody questions us about the reason for choosing one product over the other. This is a major issue faced by the people working in quality field even today as it is very difficult to decide what contributes customer loyalty or first-time sale and subsequent repeat sale. The term 'quality' means different things to different people at different times, different places and for difficult products. For example, to some users, a quality product may be one, which has no/less defects and work exactly as expected and matches with his/her concept of cost and delivery schedule along with services offered. Such a thought may be a definition of quality – '**Quality is fitness for use'**.

However, some other definitions of quality are also widely discussed. Quality defined as, '**Conformance to specifications**' is a position that people in the engineering industry often promote because they can do very little to change the design of a product and have to make a product as per the design which will best suite the user's other expectations like less cost, fast delivery and good service support. Others promote wider views, which may include the attribute of a product which satisfies /exceeds the expectations of the customer. Some believe that quality is a judgement or perception of the customer/user about the attributes of a product, as all the features may not be known or used during the entire life of a product. Quality is the extent to which the customers/users believe that the product meets or surpasses their needs and expectations. Others believe that quality means delivering product that,

- Meet customer standards, either as defined by the customer or defined by the normal usage or by some national or international bodies. (Standard may or may not be as defined by the supplier of the product. Typically for consumer goods, standard is defined by market forces and likes and dislikes of users in general.)

- Meet and fulfill customer needs which include expressed needs as well as implied requirements derived by business analysts and system analysts. Expressed needs are available in the form of requirement statement generated by users while implied needs definition may require supplier to understand customer business and provide the solution accordingly.

- One must try to meet customer expectations as maximum as possible. If something is given more than the requirements of the customer, it should be declared before transition, so that customer surprises can be

avoided. At the same time, if some aspect of the specified requirements has not been included in the product, it should be declared. This is essential so that the customer may understand the deliverables accordingly. Expectations may be at the top of customer needs may be useful in creating brand loyalty through customer delight. (Trying to get customer delight after informing customer about it.)

- Meet anticipated/unanticipated future needs and aspirations of customers by understanding their businesses and future plans. One may need to build a software and system considering some future requirements. Every product including software has a life span and due to technological inventions as well as new ways of doing things, older systems become obsolete, either technicallyor economically. How much of the future must be considered for the given product may be a responsibility of the customer or the supplier or a joint responsibility. Every product has some defined life span and may have to extrapolate future needs accordingly.

Others may simply ignore these definitions of quality and say, '**I'll know the quality of a product when I see it**'. It seems that we all '**know**' or '**feel**' somehow what the meaning of quality is, though it is very difficult to put it in words. Something that fulfils/exceeds customer's preconceived ideas about the quality is likely to be called as a quality product.

We will try to examine tools and methods which can be used to improve product quality through process approach, add value through brainstorming by producers, consumers, customers and all stakeholders about new features which may be included/old features which may be excluded from the product, decrease costs, improve schedule and help products to conform better with respect to the expressed and implied requirements. Use of quality tools and methodologies can help people engage in production related activities to improve quality of the products delivered to final user and achieve customer satisfaction.

1.4 DEFINITIONS OF QUALITY

For achieving quality of a product, one must define it insome measurable terms which can be used as a reference to find whether quality is really met or not. There are many views and definitions of quality given by stalwarts working in quality improvement and quality management arena. These definitions describe different perceptions toward quality of products. Some of these are,

1. Customer-Based Definition of Quality:

Quality product must have '**Fitness for use**' and must meet customer needs, expectations and help in achieving customer satisfaction and possibly customer delight. Any product can be considered as a quality product if it satisfies its purpose of existence through customer

satisfaction. This definition is mainly derived by an approach to quality management through '**Quality is fitness for use**'.

2. Manufacturing-Based Definition of Quality:

This definition is mainly derived from engineering product manufacturing where it is not expected that the customer knows all requirements of the product, and many product level requirements are defined by architects and designers on the basic of customer feedback/survey. Market research may have to generate requirement statement on the basis of perception of probable customers about what features and characteristics of a product are expected by the market. A quality product must have a definition of requirement specifications, design specifications, etc. and the product must conform to these specifications. The development methodologies used for the purpose must be capable of producing the right product in first go and must result into a product having no/minimum defects. This approach gives the definition of '**Conformance to requirements**'.

3. Product-Based Definitions of Quality:

The product must have something that other similar products do not have which help the customer satisfy his/her needs in a better way. These attributes must add value for the customer/user so that he/she can appreciate the product in comparison to competing products. This makes the product distinguishable from similar products in the market. Also, the customers must feel proud of at owning it due to inherent attributes and characteristics.

4. Value-Based Definition of Quality:

A product is the best combination of price and features or attributes expected by or requirements by the customers. The customer must get value for his investment by buying the product. The cost of a product has direct relationship with the value that the customer finds in it. More value for the customer helps in better appreciation of a product. Many times, it is claimed that '**People do not buy products, they buy benefits**'.

5. Transcendent Quality:

To many users/customers, it is not clear what is meant by a 'quality product', but as per their perception it is something good and they may want to purchase it because of some quality present/absent in the product. The customer will derive the value and may feel the pride of ownership.

Definitions 2, 3 and 4 are traditionally associated with the idea of a product quality that,

- A product must have zero/minimum defects so that it does not prohibit normal usage by the users. When users buy a product, they expect minimum/no failures.
- It can be purchased at a reasonable price with relation to the value users may derive from it. The customer may undertake cost benefit

analysis of the product and if benefits equal or exceed cost, it may be bought.

Customer centric product development forces the industry to look inside its own premises and thought process, making it compulsory to understand the customer. This forces the producer to create products that prospective customers may want to buy and not ones that designers think people want to receive.

The most interesting definition of quality is '**I do not know what it is, but if I'm delighted by acquiring it, I'llbuy it!**' This means that those products are better in quality which possess some characteristics that attract customers to purchase them. Though the requirements of a product may differ from customer to customer, place to place and time to time, in general, the product must be,

- Less expensive with higher returns or higher values for the customer, satisfying cost benefit analysis. Directly or indirectly, every owner would be performing cost benefit analysis before arriving at a decision to purchase something.

- Inherent of required features or attributes expected by the customer which make it fit for use. If product is of no use to users, they will never purchase it.

- Without any defect or with few defects so that its usage is uninhibited and failure or repairs would be as less as possible. If the product is very reliable, it may be liked by prospective buyers.

- With desirable cosmetic attributes.

Often, we are not aware that we want a certain product, but when we see their attributes, we feel like buying it. It may include cosmetic requirements like user interfaces, ease of use, etc.

Many of the above statements are based upon the users or customers perception about quality of the product that they wish to buy. Some of these characteristics are often attributes of products delivered by Japanese manufacturers to consumer market. Mainly, the contributors to quality would be that the failure rate are less, repairs are easier and fast, products are consistent in performance and work better than other similar products in the market and do not give any surprises to customers during use. The reasons for such improvement in the quality are a continuous/continual improvement in all aspects of product development through requirement capturing, design, development, testing, deployment and maintenance.

There is one more angle to the definition of quality of a product. Any improvement in the product quality must result into a better product, and it must give benefits to the customer finally. The benefits may be in terms of more or better features, less wait time, less cost, better service, etc. Thus, quality improvement has direct relationship with fulfilling customer requirements or giving more and more customer satisfaction.

Many people speak about not only achieving customer satisfaction but exceeding customer expectations to achieve customer delight. There is a signal of caution about exceeding customer expectations. Any feature which surprises a customer may not be appreciated by him/her and may be termed as a defect. Exceeding expectations without informing the customer about what they can expect in addition to defined requirements can be dangerous and may result in rejection of such products.

1.5 CORE COMPONENTS OF QUALITY

Quality of a product must be driven by customer requirements and expectations from the product. Those expectations may be expressed as a part of requirement specifications defined or may be implied one which is generally accepted as requirements. It must have some important characteristics that may help customer in getting more and more benefits and satisfaction by using the product. Some postulates of quality are.

1.5.1 Quality Is Based on Customer Satisfaction By Acquiring A Product:

Quality is something perceived by a customer while using a product. The effort of a quality product, delivered and used by a customer, on his satisfaction and delight is the most important factor in determining whether the quality has been achieved or not. It talks about the ability of a product or service to satisfy a customer by fulfilling his needs or purpose for which it is acquired. It may come through the attributes of a product, time required for a customer to acquire it, price a customer is expected to pay for it and so many other factors associated with the product as well as the organisation producing or distributing it. All these factors may or may not be governed by the manufacturer alone but may be dependent on quality of inputs. This point stresses a need that a producer must understand the purpose or usage of a product and then devise a quality plan for it accordingly, to satisfy the purpose of the product.

1.5.2 The Organisation Must Define Quality Parameters Before It Can Be Achieved:

We have already discussed that quality is a perception of a customer about satisfaction of needs or expectation. It is difficult for the manufacturer to achieve the quality of product without knowing what customer is looking for while purchasing it. If product quality is defined in some measurable terms, it can help the manufacturer in deciding whether the product quality has been achieved or not during its manufacturing and delivery. In order to meet some criteria of improvement and ability to satisfy a customer, one must follow a cycle of ‘Define’, ‘Measure’, ‘Monitor’, ‘Control’ and ‘Improve’. The cycle of improvements through measurements is described below,

Define:

There must be some definition of what is required in the product, in terms of attributes or characteristics of a product, and in how much quantity it is

required to derive customer satisfaction. Features, functionalities and attributes of the product must be measured in quantitative terms, and it must be a part of requirement specification as well as acceptance criteria defined for it. The supplier as well as the customer must know what 'Must be', what 'Should be' and what 'Could be' present in the product so delivered and also what 'Must not be', 'Should not be' and 'Could not be' present in the product.

Measure:

The quantitative measures must be defined as an attribute of quality of a product. Presence or absence of these attributes in required quantities acts as an indicator of product quality achievement. Measurement also gives a gap between what is expected by a customer and what is delivered to him when the product is sold. This gap may be considered as a lack of quality for that product. This may cause customer dissatisfaction or rejection by the customer.

Monitor:

Ability of the product to satisfy customer expectations defines the quality of a product. There must be some mechanism available with the manufacturer to monitor the processes used in development, testing and delivering a product to a customer and their outcome, i.e., attributes of product produced using the processes, to ensure that customer satisfaction is incorporated in the deliverables given to the customer. Derivations from the specifications must be analysed and reasons of these derivations must be sorted out to improve product and process used for producing it. An organisation must have correction as well as corrective and preventive action plans to remove the reasons of deviations/deficiencies in the product as well as improve the processes used for making it.

Control:

Control gives the ability to provide desired results and avoid the undesired things going to a customer. Controlling function in the organisation, properly called as 'quality control' or 'verification and validation', may be given a responsibility to control product quality at micro level while the final responsibility of overall organisational control is entrusted with the management, popularly called as 'quality assurance'. Management must put some mechanism in place for reviewing and controlling the progress of product development and testing, initiating actions on deviations/deficiencies observed in the product as well as the process.

Improve:

Continuous/continual improvements are necessary to maintain ongoing customer satisfaction and overcome the possible competition, customer complaints, etc. If some producer enjoys very high customer satisfaction and huge demand for his product, competitors will try to enter the market. They will try to improve their products further to beat the competition.

Improvement may be either of two different approaches viz. continuous improvement and continual improvement as the case may be.

1.5.3 Management Must Lead The Organisation Through Improvement Efforts:

Quality must be perceived by a customer to realise customer satisfaction. Many factors must be controlled by a manufacturer in order to attain customer satisfaction. Management is the single strongest force existing in an organisation to make the changes as expected by a customer; it naturally becomes the leader in achieving customer satisfaction, quality of product and improvement of the processes used through various programs of continuous/continual improvement. Quality management must be driven by the management and participated by all employees.

Management should lead the endeavour of quality improvement program in the organisation by defining vision, mission, policies, objectives, strategies, goals and values for the organisation and show the existence of the same in the organisation by self-examples. Entire organisation should initiate the behavioural and leadership aspects of the management. Every word and action by the management may be seen and adopted by the employees. Quality improvement is also termed as a 'cultural change brought in by management'.

Organisation based policies, procedures, methods, standards, systems etc. are defined and approved by the management. Adherence to these systems must be monitored continuously and deviation/deficiencies must be tracked. Actions resulting from the observed deviations/deficiencies shall be viewed as the areas which need improvements. The improvements may be required in enforcement or definition of policies and procedures, methods, standards, etc. Management must have quality planning at organisation level to support improvement actions.

1.5.4 Continuous Process (Continual) Improvement is Necessary:

There was an old belief that quality can be improved by more inspection, testing and rework, scrap, sorting, etc. it was expected that a customer must inspect the product and report the defects as and when they were reported by the customer. This added to the cost of inspection, segregation, failure, rework, etc. for the customer and reduced the profit margins for the manufacturer or increased the price for the customer. At the same time, customer's right to receive a good product was withdrawn.

For improving the competitive cost advantage to producer as well as customer, quality must be produced with an aim of first time right and must be improved continuously/ continually. For the customer, total cost product is inclusive of cost of purchase and maintenance. Total cost is more important to him than the purchase tag. The first step for producing quality is the definition of processes used for producing the product and the cycle of continuous or continual improvement (Plan-Do-Check-Act or Define-Measure-Monitor-Control-Improve) to refine and redefine processes to achieve targeted improvements. It needs 'Planning' for

quality, 'Doing' as per the defined plans, 'Checking' the outcome at each stage with expected results and taking 'Actions' on the variances produced. Refer Table 1.1 for comparison between continuous and continual improvement.

Table 1.1 Comparison of continuous and continual improvement

Continuous improvement	Continual improvement
Continuous improvement is dynamic in nature. The changes are done at every stage and every time to improve further.	Continual improvement is dynamic as well as static change management. The changes are done, absorbed, baselined and sustained before taking next step of improvements.
Continuously striving for excellence gives a continuous improvement	Periodic improvements followed by stabilisation process and sustenance represent continual improvement
It has a thrust on continuous refinement of the processes to eliminate waste continuously	Stabilisation of processes at each iteration of improvement where waste is removed in stages
It has high dependence on people having innovative skills tending towards inventions	Less dependence on people and more dependence on innovation processes
Environment is continuously changed	Changes in environment are followed by stabilisation
Sometimes it creates a turbulence in an organisation, if people are not able to digest continuous change	It may be better suited than continuous improvement. It gives a chance to settle the change before next change is introduced

1.6 QUALITY VIEW

Stakeholders are the people or entities interested in success / failure of a project or product or organisation in general. Every product / project / organisation has several stakeholders interested in its betterment. Quality is viewed differently by different stakeholders of the product / project / organisation as per their role in entire spectrum. Some quality models put all stakeholders for the project and product in six major categories. These stakeholders benefit directly or indirectly, if the project / product / organization becomes successful, and suffer, if the organisation or project fails.

Customer:

Customer is the main stakeholder for any product/project. The customer will be paying for the product to satisfy his requirements. He/she must benefit by acquiring a new product. Sometimes, the customer and user can be different entities but here, we are defining both as same entity considering customer as a user. Though sometimes late delivery penalty clauses are included in contract, the customer is interested in the product delivery with all features on defined scheduled time and may not be interested in getting compensated for the failures or delayed deliveries.

Supplier:

Suppliers give inputs for making a project/product. As an organisation becomes successful, more and more projects are executed, and suppliers can make more business, profit and expansion. Suppliers can be external or internal to the organisation. External suppliers may include people supplying machines, hardware, software, etc. for money while internal suppliers may include other functions such as system administrator, training provider, etc, which are supporting projects/product development.

Employee:

People working in a project/an organisation may be termed as employees. These people may be permanent temporary workers but may not be contractual labours having no stake in product success. (Contractual workers may come under supplier category.) As the projects organisations become successful, people working on these projects in these organisations get more recognition, satisfaction, pride, etc. They feel proud to be part of a successful mission.

Management:

People managing the organization / project may be termed as management in general. Management may be divided further into project management, staff management, senior management, investors, etc. Management needs more profit, recognition, turnover improvements, etc to make their vision and mission successful. Successful projects give management many benefits like expanding customer base, getting recognition, more profit, more business, etc.

There are two more stakeholders in the success as well as failure of any project / product / organisation. Many times, we do not feel their existence at project level or even at organisation level. But they do exist at macro level.

Society:

Society benefits as well as suffers due to successful projects /organisations. It is more of a perception of an individual looking towards the success of the organisation. Successful organisations / projects generate more employment, and wealth for the people who are in the category of customer, supplier, employee, management, etc. It also affects the resource availability at local as well as global level like water, roads, power supply, etc. It also affects economics of a society to a larger extent. Major price rise has been seen in industry dominated areas as the paying capacity of people in these areas is higher than other areaswhere there is no such industry.

Government:

Government may be further categorised as local government, state government, central government, etc. Government benefits as well as suffers due to successful projects/organisations. Government may get higher taxes, export benefits, foreign currency, etc, from successful projects organisations, People living in those areas may get employment and overall wealth of the nation improves. At the same time, there may be pressure on resources like water, power, etc. There may be some problems in terms of money availability and flow as success leads to more buying power and inflation.

Quality perspective of all these stakeholders defines their expectations from organisation projects. We may feel that superficially these views

may differ from each other though finally they may be leading to the same outcome, If these views match perfectly and there is no gap in the stakeholder's expectations, then organisational performance and effectiveness can he improved significantly as collective efforts from all stakeholders, If the views differ significantly, this may lead to discord and hamper improvement. Let us discuss two important views of quality which mainly defines the expectations from a project and success of a project at unit level viz. customer's view and developer's view of quality.

1.6.1 Customer's View of Quality:

Customer's view of quality of product interprets customer requirements and expectation for getting a better product at defined schedule, cost and with adequate service alone with required features and functionalities. Customer is paying some cost to get a product because he finds value in such acquisition.

Delivering Right Product:

The products received by customers must be useful to satisfy their needs and expectations. It may or may not be the correct product from manufacturer's perspective or what business analyst/system designer may think. There is a possibility that development team including testers may ask several queries about the requirements of the product to get them clarified but the final decision about the requirements definition shall be with customer. If customer confirms that the requirements are correct not correct, then there is no possibility of further argument about the validity/invalidity of requirements.

Satisfying Customer's Needs:

The product may or may not be the best product, as per the manufacturer's views or those which are available in the market, which can be made from the given set of requirements and constraints. There are possibilities of different alternatives for overcoming the constraints and implementing the requirements, Basic constraint in product development and testing is that product must be capable of satisfying customer needs. Needs are ‘must’ among requirements from customer's perspective. They may be part of processes for doing requirement analysis and selection of approach for designing on the basis of decision analysis and resolution, using some techniques such as cost-benefit analysis, etc. which suites best in the given situation. This must help organisation to achieve customer satisfaction through product development and delivery.

Meeting Customer Expectations:

Customer expectations may be categorised into two parts viz. expressed expectations and implied expectations. Expectations documented and given formally by the customer are termed as ‘expressed requirements’ while ‘implied expectations’ are those, which may not forma part of requirement specifications formally but something which is expected by customer by default. It is a responsibility of a developing organisation to

convert, as many as possible, implied requirements into expressed requirements by asking queries or eliciting requirements. One must target for 100°a conversion of implied requirements into expressed requirements, though difficult, as developer may refuse to accept the defect belonging to implied requirements simply because it is not a part of requirement statement and they may not be aware of such things.

Treating Every Customer with Integrity, Courtesy, and Respect:

Customer and the requirements assessed (both expressed as well as implied) are very important for a developing organisation as customer will be paying on the basis of value he finds in the product. Definition of the requirements is a first step to satisfy the customer through '**Conformance to Requirements**'. Requirements may be documented by anybody, generally development team, but customer is the owner of the requirements. Organisation shall believe and understand that the customer understands what is required by him. How to achieve these requirements is a responsibility of a developing organisation. He may be given suggestions, sharing some past experiences and knowledge gained in similar projects but manufacturer shall not define requirements or thrust or push the requirements to customer. It is quite often quoted that the customer does not know or understand his requirements. This opinion cannot be bought by anybody.

Customer telephone calls and mails must be answered with courtesy and in reasonable time. The information provided to the customer must be accurate and he/she must be able to depend on this information. The customer is not a hindrance to the project development but he is the purpose of the business and producer must understand this.

1.6.2 Supplier's View of Quality:

Supplier is a development organisation in the context of software application development, Supplier has some expectations or needs, which must be satisfied by producing a product and selling it to customer. Supplier expectations may range from profitability, name in market, repeat orders, customer satisfaction, etc. These expectations may be fulfilled in the following ways,

Doing the Right Things:

Supplier is intended to do right things for the fir time so that there is no waste, scrap, rework, etc. Wastes like rework, scrap, and repairs are produced by hidden factory for which there is no customer, Changes in requirements are considered as problems during product development, if supplier is expected to absorb the costs associated with it. Changes in requirement cause rework of design, development, testing etc. This adds to the cost of development but if the price remains same, it is a loss for manufacturer. It also adds to fatigue and frustration of the people involved in development as they find no value in reworking, sorting, scrapping, etc. Following right processes to get the product required by the customer and achieving customer satisfaction and profits as well as job satisfaction may

be the expectations of a supplier. Suppliers may require clear and correct definition of deliverables, time schedules, attributes of product, etc.

Doing It the Right Way:

A producer may have his own methods, standards and processes to achieve the desired outputs. Sometimes, customer may impose the processes defined by him for building the product on the producer. Ability of development process to produce the product as required by the customer defines the capability of development process. These process definitions may be an outcome of quality standards or models or business models adopted by the supplier or customer. As organisation matures, the processes towards excellence- by refining and redefining processes and process capability- must improve continually/continuously. As the processes reach optimisation, they must result into better quality products and more satisfaction for the customer and also fulfill vendor requirements.

Doing it Right the First Time:

Doing right things at the first time may avoid frustration, scrap, rework, etc. and improve profitability, reduce cost and improve customer satisfaction for the supplier. Doing right things at the first time improves performance, productivity and efficiency of a manufacturing process. This directly helps in improving profitability and gives advantage in a competitive market. Supplier would always like to follow the capable processes which can get the product right at the first attempt.

Doing It on Time:

All resources for developing a new product are scarce and time factor has a cost associated with it. The value of money changes as time changes. Thus, money received late is as good as less money received. If the customer is expected to pay on each milestone, then the producer has to deliver milestones on time to realise money on time. Delay in delivery represents a problem with processes, starting from getting requirements, estimation of efforts and schedule and goes up to delivery and development.

Difference between the two views discussed above (Customer's view vs Supplier's view) creates problem for manufacturer as Well as customer when it comes to requirement gathering, acceptance testing, etc. It may result into mismatch of expectation and service level, which would be responsible for introducing defects in the product in terms of functionalities, features, cost, delivery schedule, etc. Sometimes these views are considered as two opposite sides of a coin or two poles of the Earth which can never match. The difference between two views is treated as a gap.

In many cases, the customer wants to dictate the terms as he is going to pay for the product and the processes used for making it, while the supplier wants customer to accept whatever is produced. Wherever the gap

is less, the ability of a product to satisfy customer needs is considered better. At the same time, it helps a development organisation as well as a customer to get their parts of benefits from steady processes. On the contrary, larger gap causes more problems in development, distorted relations between two parties and may result into loss for both. Quality processes must reduce the gap between two views effectively. Effectiveness of a quality process may be defined as the ability of the processes and the product to satisfy both or all stakeholders in achieving their expectations.

Figure 1.1 explains a gap between actual product, requirements for the product and customer expectations from the product. It gives two types of gaps, viz. user's gap and producer's gap.

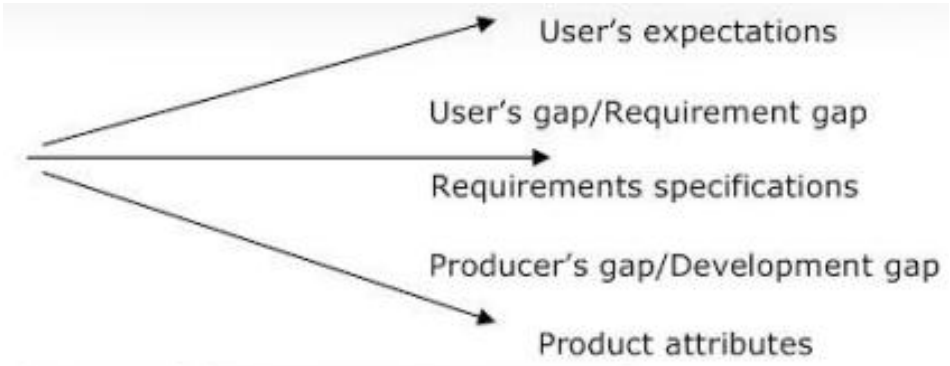


Fig. 1.1 User's gap and Producer's gap

1.6.3 User's Gap/Requirement Gap:

User's gap is a gap between requirement specifications for the product and user expectations from it. This gap focuses on the difference in the final product attributes as defined by requirement dements with respect to the intents of the user. Developers must convert user needs into the product requirement specifications and create the product exactly as per these specifications. This may need understanding and interpretation of customer's business flow and requirements, and how the product is intended to be used by the customer to satisfy his business requirements.

Closing User's Gap:

User's gap represents the failure seen by the customer in terms of difference between user needs and product specifications. An organisation must apply some processes and methods so that user's gap can be closed effectively or reduced to as little as possible, Methods for closing these gaps may depend on organisation's way of thinking, resource availability, type of customer, customer's thought process, etc. Some of these methods are mentioned below.

Customer Survey:

Customer surveys are essential when an organisation is producing a product for a larger market where a mismatch between the product and

expectation can be a major problem to producer. It may also be essential for projects undertaken for a single customer such as mission critical projects where failure of the project has substantial impact on the customer as well as producer. For a larger market, survey may be conducted by marketing function or business analyst to understand user requirements and collate them into specification documents for the product. Survey teams decide present and future requirements for the product and the features required by the potential customers.

For a single customer, a survey is conducted by business analyst and system analyst to analyse the intended use of the product under development and the possible environment of usage. Domain experts from the development side may visit the customer organisation to understand the specific requirements and work flow related to domain to incorporate it into the product to be developed.

Joint Application Development (JAD):

Applications are developed jointly by customer and manufacturer where there is close co-ordination between two teams. In joint application development, users or customers may be overseeing the system development and closely monitor requirements specifications, architecture, designs, coding, testing and test results. The applications produced by this method may follow top-down approach where user interfaces and framework are developed first and approved by the users and then logic is built behind it. Some people may call joint application development as an agile methodology of development where developers collaborate with customer during development.

User Involvement in Application Development:

This approach works on the similar lines of joint application development (JAD). User may be involved in approving requirement specifications, design specification, application user interfaces, etc. He/she may have to answer the queries asked by developers and provide clarifications, if any. An organisation may develop a prototype, model, etc, to understand user requirements and get approval from the user team. If the organisation is producing products for a larger market, few representative users may be considered for collecting requirement specifications, test case designing and acceptance testing of the application under development.

1.6.4 Producer's Gap/Development Gap:

Producer's gap is a gap between product actually delivered and the requirement and design specifications developed for the product. The requirement specifications written by business analyst may not be understood in the same way by the development team. There are communication losses at each stage and business analyst and developers/testers may not be located next to each other to provide explanation for each and every requirement. More stages of communication may lead to more gaps and more distortion of

requirements. The product so produced and requirement specifications used may differ significantly creating producer's gap

Closing Producer's Gap:

Producer's gap represents a failure on part of development team to convert requirements into product. Producer's gap can be seen as the defects found in in-house testing. Producer's gap is due to process failure at producer's place and there must he process improvement plans to close this gap

Process Definition:

Development and testing processes must be sufficiently mature to handle the transfer of information from one person or one stage to another during software development life cycle. There must be continuous " Do' and Check" processes to build better products and get feedback about the process performance. Such product development has lifecycle testing activities associated with development.

Work Product Review:

As the stages of software development life cycle progresses, one may have to keep a close watch on artifacts produced during each stage to find if any inconsistency has been introduced with respect to earlier phase. Generation of requirement traceability matrix is an important factor in this approach.

1.7 FINANCIAL ASPECT OF QUALITY

Earlier, people were of the opinion that more price of a product represents better quality as it involves more inspection, testing, sorting, etc, and ensures that only good parts are supplied to the customer, Sales price was defined as,

Sales price = Cost of manufacturing + Cost of Quality + Profit

If we consider the monopoly way of life, this approach may be considered good since the price is decided by the manufacturer depending upon three factors described above. Unfortunately, monopoly does not exist in real world. If any product enjoys higher profitability, more number of producers would enter into competition.

Number of sellers may exceed number of buyers. When the products produced match in all aspects, the cost would decide the quantity of sale. The competitor who reduces the price, may get more volume of sale if all other things remain constant. Reducing the sales price reduces percentage profit. For maintaining profit, the producer may try to reduce cost of production without compromising on the quality aspect.

Thus, in a competitive environment, the equation changes to

Profit = Sales price - [Cost of manufacturing + Cost of Quality]

1.7.1 Cost of Manufacturing:

Cost of manufacturing is a cost required for developing the right product by right method at the first time. The money involved in resources like material, people, licenses, etc. forms a cost of manufacturing. The cost of manufacturing remains constant over the lime span for the given project and given technology and it has a direct relationship with the efforts. It can be reduced through improvements in technology and productivity but it may need longer time frame. The cost involved in requirement analysis, designing, development and coding are the casts associated with manufacturing.

1.7.2 Cost of Quality:

Cost of quality represents the part of cost of production incurred in improving or maintaining quality of a product. Some people keep cost of manufacturing and cost of quality as separate while others may include them under cost of production. Cost of manufacturing may be supported by cost of quality and there exists anointer relationship between the two costs.

Cost of quality includes all the efforts and cost incurred in prevention of defects, appraisal of product to find whether it is suitable to customer or not and fixing of defects or failures at various levels as and when they are reported and conducting any retesting, regression testing, etc.

Cost of Prevention:

An organisation may have defined processes, guidelines, standards of development, testing, etc. It may define a program of imparting training to all people involved in development and testing. This may represent a cost of prevention. Creation and use of formats, templates, etc. acquiring various process models and standards, etc, also represent a cost of prevention. This is an investment by an organisation and it is supposed to yield returns. This is also termed as '**Green money**'**.** Generally, it is believed that 1 part of cost of prevention can reduce 10 parts of cost of appraisal and 100parts of cost of failure.

Cost of Appraisal:

An organisation may perform various levels of reviews and testing to appraise the equality of the product and the process followed for developing the product. The cost incurred in first time reviews and testing is called as the cost of appraisal. There is no return on investment but this helps in identifying the process capabilities and process related problems, if any. This is termed as '**Blue money**' as it can be recovered from the customer. Generally, it is believed that 1 part of cost of appraisal can reduce 10parts of cost of failure.

Cost of Failure:

Cost of failure starts when there is any defect or violation detected at any stage of development including post-delivery efforts spent on defect

fixing. Any extent of rework, retesting, sorting, scrapping, regression testing, late payments, sales under concession, etc. represents cost of failure. There may be some indirect costs such as loss of goodwill, not getting customer references, not getting repeat orders, and customer dissatisfaction associated with the failure to produce the right product. The cost incurred due to some kind of failure is represented as cost of failure. This is termed as **'Red money'**. This cost affects the profitability of the project organisation badly.

On the basis of quality focus, organisations may be placed in 2 categories viz, organisations which are less quality conscious (termed as ‘q’) and organisations which are more quality conscious (termed as 'Q'). The conscious (termed as 'q') and organisations which are more quality conscious (termed as 'Q'). The distribution of cost of quality for these two types may be represented as below. Please refer Fig. 1.2

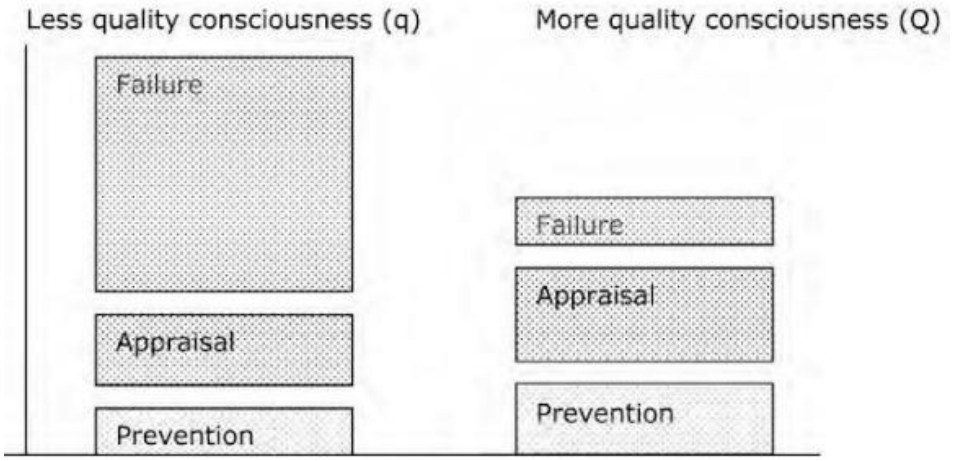


Fig. 1.2 Cost of Quality

The bottommost rectangle represents cost of prevention. As the organisation's quality consciousness increases, prevention cost increases substantially due to introduction of various process requirements. The organization may have defined processes, methods, work instructions, standards, guidelines, templates, formats etc. Teams are supposed to use them while building a product or testing it, and conduct audits which need resources, time and money. Project teams may create project plan, test plan, assess the risks and issues faced during development, decide on the actions to reduce their probabilities/impacts, etc. More cost of prevention may be justifiable for a project/product only if it reduces the cost of failure.

Middle rectangle represents cost of appraisal. As quality consciousness increases, cost of appraisal also increases. An organisation may prepare various plans (such as quality plan and test plan) and then review these plans, write the test cases, define test data, execute test cases, analyse defects and initiate actions to prevent defect recurrence. There may be checklists, guidelines, etc. used for verification and validation activities. The cost incurred in appraisal must be justifiable and must reduce cost of failure.

The topmost rectangle represents cost of failure, Cost of failure includes the cost associated with any failure which may range from rework, retesting, etc. including customer dissatisfaction or loss of revenue, etc.

As quality consciousness improves, failure cost must reduce representing better quality of products offered and higher customer satisfaction. Thus, the overall cost of quality must reduce as quality consciousness of the organisation increases.

1.8 DEFINITION OF QUALITY

Let us try to redefine and understand the meaning of the term 'Quality' with a new perspective. Many definitions of the word 'Quality' are available and are used at different forums by different people. Most of these definitions show some aspect of quality. While no definition is completely wrong, no definition is completely right also. Few of the definitions prescribed for quality are,

Predictable Degree of Uniformity, Dependability at Low Cost and Suited to Market:

This definition stresses on quality as an attribute of product which is predictable and uniform in behaviour throughout product usage and stresses on the ability of a product to give consistent results again and again. One must be able to predict the product behaviour beforehand. It talks about reduction in variability of a product in terms of features, performance, etc. It also defines the dependability aspect of quality product. One must expect reliable results from quality products every time they are used. Quality product must be the cheapest one as it talks about reducing failure cost of quality hike rework, scrap, sorting, etc. The most important thing about product quality is that it must help the product to suite the market needs or expectations. This necessitates that the manufacturer should produce those products which can be sold in the market.

Degree to Which a Set of Inherent Characteristics of the Product / Service Fulfils the Requirements:

This definition of quality stresses the need that the product must conform to defined and documented requirement statement as well as expectations of users, Higher degree of fulfilments of these requirements makes a product better in terms of quality. There must be something in the product, defined as attributes of product', which ensures customer satisfaction. The attributes must be inborn in the product, indicating capable processes used for developing such a product.

Ability of a Product or Service That Bears Upon Its Ability to Satisfy Implied or Expressed Need:

This definition of quality of product is derived from the approach of '**Fitness for use**', It talks about a product achieving defined requirements as well as implied requirements, satisfying needs of customer for which it

is being used. Better ability of a product to fulfill requirements makes a product better, from quality perspective.

1.9 CUSTOMERS, SUPPLIERS AND PROCESSES

For any organisation, there are some suppliers supplying the inputs required and some customers who will be buying the outputs produced. Suppliers and customers may be internal or external to the organisation. In the larger canvas, an entire organisation can be viewed as the component in a huge supply chain of the world where products are made by converting some inputs which may act as inputs to the next stage. External suppliers provide input to the organisation and external customer receive the output of the organisation. In turn, suppliers may be customers for some other organisations and customer may be acting as suppliers for somebody else down the line.

Internal Customer:

Internal customers are the functions and projects serviced and supported by some other functions projects. System administration may have projects as their customer while purchasing may have system administration as their customer. During value chain, each function must understand its customers and suppliers. Each function must try to fulfill its customer requirements. This is one of the important considerations behind 'Total Quality Management' where each and every individual in supplychain must identify and support hi customer, If internal customers are satisfied, this will automatically satisfy external customer as it sets the tone and perspective for everybody.

External Customer:

External customers are the external people to the organisation who will be paying for the services offered by the organisation. These are the people who will be actually buying products from the organisation. As the organisation concentrates on external customer for their satisfaction, it must improve equality of its output.

1.10 TOTAL QUALITY MANAGEMENT (TQM)

'**Total quality management**' principle intends to view internal and external customers as well as internal and external suppliers for each process, project and for entire organisation as a whole. The process and factions of an organisation can be broken down into component elements, which act as suppliers/customers to each other during the work flow. Each supplier eventually also becomes a customer at some other moment and vice-versa. If one can take care of his/her customer (whether internal or external) with an intention to satisfy him, it may result into customer satisfaction and continual improvement for the organisation.

Supply chain relationship may be defined graphically as, shown in Fig. 1.3

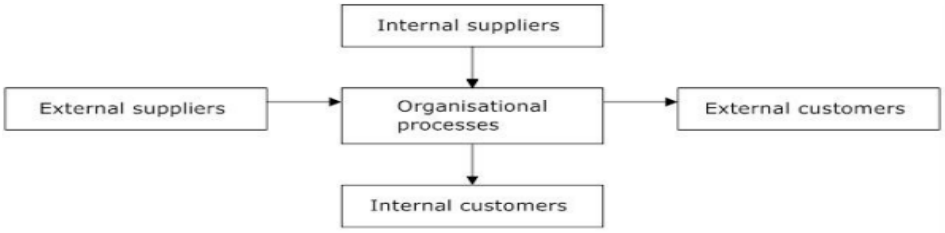


Fig. 1.3 Supply chain relationship between suppliers and customers

'Total quality management' (TQM) is the application of quality principles to all facets and business processes of an organisation. It talks about applying quality methods to the entire organisation whether a given function or part of the organisation faces external customer(s) or not. One clear definition of quality involves satisfying one's customer irrespective of whether he/she is outside or inside the organisation. This implementation of customer satisfaction has different meanings for different parts of an organisation.

Quality Management Approach:

Dr Edward Deming implemented quality management system driven by '**Total Quality Management**' and '**Continual Improvement**' in Japanese environment. It resulted into repetitive, cost-effective processes with an intention to satisfy customer requirements and achieve customer satisfaction. Implementation was inclined toward assessment of quality management system which was adoptive to the utility of tools for understanding data produced by the process measurements, Dr Deming proposed principles for quality management that are widely used by the quality practitioners.

1.11 QUALITY PRINCIPLES OF 'TOTAL QUALITY MANAGEMENT

'Total quality management' works on some basic principles of quality management definition and implementation. These have evolved over a span of experimentation and deployment of quality culture in organisations.

1.11.1 Develop Constancy of Purpose of Definition and Deployment of Various Initiatives:

Management must create constancy of purpose for products and processes, allocating resources adequately to provide for long term as well as short term needs rather than concentrating on short term profitability: suppliers and organisation must have an intent to become competitive in the world, to stay in business and to provide jobs to people and welfare of the society. The processes followed during entire lifecycle at product development from requirement capturing till final delivery must be consistent with each other and must be followed over a larger horizon, Decisions taken by management at different instances must be consistent and based upon same standards, rules and regulations. Different initiatives

by management must have relationships with each other and must be able to satisfy the vision of the organisation.

1.11.2 Adapting to New Philosophy of Managing People/Stakeholders by Building Confidence and Relationships:

Management must adapt to the new philosophies of doing work and getting the work done from its people and suppliers. We are in a new economic era where skills make an individual indispensable. The process started in Japan and is perceived as a model throughout the world for improving quality of working as well as products.

We can no longer live with commonly accepted levels of delays, mistakes, defective materials, rejections and poor workmanship. Transformation of management style to total quality management is necessary to take the business and industry on the path of continued improvements.

1.11.3 Declare Freedom from Mass Inspection of Incoming/Produced Output:

It was a common belief earlier that for improving quality of a product, one needs to have rigorous inspection program followed by huge rework, sorting, scrapping, etc. It was believed that one must check everything to ensure that no defect goes to customer. But there is a need for change in thinking of management and people as mass inspection results into huge cost overrun and product produced is of inferior quality. There must be an approach for elimination of mass inspection followed by cost of failure as the way to achieve quality of products. Improving quality of products requires setting up the right processes of development and measurement of process capabilities and statistical evidence of built-in quality in all departments and functions.

1.11.4 Stop Awarding of Lowest Price Tag Contracts to Suppliers:

Organisations must end the practice of comparing unit purchase price as a criterion of awarding contracts. Vendor selection must be done on the basis of total cost including price, rejections, etc. Organisations must perform measurements of quality of supply along with price and do the source selection on the basis of final cost paid by it in terms of procurement, rework, maintenance, operations, etc. It must reduce the number of suppliers by eliminating those suppliers that do not qualify with statistical evidences of quality of their supply. This will automatically reduce variation and improve consistency. Aim of vendor selection is to minimise total cost, not merely initial cost of purchasing, by minimising variations in the vendor supplied product. This may be achieved by moving towards lesser/single supplier, on a long-term relationship of loyalty and trust. Organisations must install statistical process control in development even at vendor site and reduce the variation in place of inspecting each and every piece.

1.11.5 Improve Every Process Used for Development and Testing of Product:

Improve every process of planning, production and service to the customer and other support processes constantly. Processes have interrelationships with each other and one process improvement affects other processes positively. Search continually for problems in these processes in terms of variations in order to improve every activity in the organisation- to improve quality and productivity and decrease the cost of production as well as cost of quality continuously. Institutionalise innovations and improvements of products and processes used to make them. It is management's job to work continually for optimising processes.

1.11.6 Institutionalise Training Across the Organisation for all People:

An organisation must institute modem methods of training which may include on-the-job training, classroom training, self-study, etc. for all people, including management, to make better use of their abilities. New skills are required to keep up with the changes in materials, methods, product and service design, infrastructure,

techniques and service Skill levels of people can be enhanced to make them suitable for better performance by planning different training programs, Training technical as well as nontechnical should focus on improving present skills and acquiring new skills Customer requirements may also change and people may need training to understand changes in customer expectations, Suppliers may he included in training programs to derive better output from them.

1.11.7 Institutionalise Leadership Throughout Organisationat Each Level:

An organisation must adopt and institute leadership at all levels with the aim of helping people to do their jobs in a better way. Responsibility of managers and supervisors must be changed from controlling people to mentoring, guiding, and supporting them. Also, their focus should shift from number of work items to be produced to quality of output. Improvement in quality will automatically improve productivity by reducing `` scrap, inspection, rework, etc. Management must ensure that immediate actions are taken on reports of inherited defects, maintenance requirements, poor tools, fuzzy operational definitions and all conditions detrimental to quality of products and services offered to final users,

1.11.8 Drive Out Fear of Failure from Employees:

An organisation must encourage elective two-way communication and other means to drive out fear of failure from the minds of all employees. Employees can work effectively and more productively to achieve better quality output when there is no fear of failure, People may not try new things if they are punished for failure.

Management should not stop or discount feedback coming to them even if it is negative. Giving positive as well as negative feedback should be encouraged, and it must be used to perform SWOT analysis followed by actions. Fear is something which creates stress in minds of people, prohibiting them from working on new ways of doing Things, Fear can cause disruption in decision-taking process which may result into excessive defense and also, major defects in the product. People may not be able to perform under stress.

1.11.9 Break Down Barriers Between Functions/Departments:

Physical as well as psychological breaking down of barriers between departments and staff areas may create a force of cohesion, People start performing as a lam and there is synergy of group activities. People in different areas must work as a team to tackle problems that may be encountered with products and customer satisfaction, Ultimate aim of the organisation must be to satisfy customers. All people working together and helping each other to solve the problems faced by customers can help in achieving this ultimate aim.

1.11.10 Eliminate Exhortations By Numbers, Goals, Targets:

Eliminate use of slogans, posters and exhortations of the work force, demanding ‘Zero Defects’ and new levels of productivity, without providing methods and guidance about how to achieve it. Such exhortations create adverse relationships between supervisors and workers. Main cause of low quality and productivity is in the processes used for production as the numbers to be achieved may be beyond the capability of the processes followed by the workers. This may induce undue frustration and stress and may lead to failures. The Organisation shall have methods to demonstrate that the targets can be achieved with smart work.

1.11.11 Eliminate Arbitrary Numerical Targets Which Are Not supported By Processes:

Eliminate work standards that prescribe quotas for the work force and numerical goals for managers to be achieved. Substitute the quotas with mentoring and support to people, and helpful leadership in order to achieve continual improvement in quality and productivity of the processes, Numerical goals should not become the definition of achievement/targets. There must be a methodology to define what achievement is and what must be considered as a stretched target.

1.11.12 Permit Pride of Workmanship for Employees:

Remove the barriers that take away the pride of workmanship for workers and management. People must feel proud of the work they are doing, and know how they are contributing to organisational vision. This implies complete abolition of the annual appraisal of performance and of ‘management by objective’. Responsibility of managers and supervisors

must be changed from sheer numbers to quality of output. Management must understand managing by facts.

1.11.13 Encourage Education of New Skills and Techniques:

Institute a rigorous program of education and training for people working in different areas and encourages lf-improvement programs for everyone. What an organisation needs is not just good people; it needs people who can improve themselves with education to accept new challenges. Advances in competitive position will have their roots in knowledge gained by people during such trainings.

1.11.14 Top Management Commitment and Action to Improve Continually:

Clearly define top management's, commitment to ever-improving quality and productivity and their obligation to implement quality principles throughout the organisation. It is not sufficient that the top management commits for quality and productivity but employees must also see and perceive their commitment. They must know what it is that they are committed to i.e., what they must do in order to show their commitment.

1.12 QUALITY MANAGEMENT THROUGH STATISTICAL PROCESS CONTROL

Dr Joseph Juran is a pioneer of statistical quality control with a definition of improvement cycle through Define, Measure, Monitor, Control and Improve (DMMCI). One must understand the interrelationships among customers, suppliers and processes used in development, testing, etc, and establish quality management based on metrics program. There are three parts of the approach, namely,

1.12.1 Quality Planning At All Levels:

Quality is not an accident but is a result of deliberate effort to achieve something which is defined in advance. An organisation must have a definition of what they wish to achieve Quality improvement must be planned at all levels of organisation and then only it can be achieved. Quality planning happens at two levels viz.organisation level and individual department function project level.

Quality Planning at Organisation Level:

Quality must he planned at organisation level first. It must be in the form of policy definition and strategic quality plans on the basis of vision, missions) and policies set by senior management. Planning process must attempt to discover who the customers are at present and who will be the customers in future, and what are and will be their needs and expectations from the organisation. The needs of the present or future customers must be expressed in numeric term so that the actions can be planned and progress can be measured. The presence or absence of different attributes

to the extent required shall define the quality level of the product or services.

Quality Planning at Unit Level:

Quality planning at unit level must be done by the people responsible for managing the unit. Operational quality plans must be in sync with organisational policies and strategies. Project plan and quality plan at unit level must be consistent with the strategic quality plans at organisation level and must be derived from the organisation's vision and mission(s).

1.12.2 Quality Control:

Quality control process attempts to examine the present product at various levels with the defined standards so that an organisation may appraise the outcome of the processes. Removing defects in the processes to improve their capability can help to reach new levels of improved quality. (e.g., process improvement must be targeted towards lowering defects, reducing cost, and improving customer satisfaction.) It must measure the deviations with respect to the number of achievements planned in quality planning so that the organization can initiate the actions to reduce the deviations to minimum level.

1.12.3 Quality Improvement:

Improvement process attempts to continuously improve the quality of the process used for producing products. Quality of the process is measured on the basis of the attributes of the products produced. There isno end to quality improvements and it needs to take newer challenges again and again. Finding deviations in the attributes of products and processes with respect to planned levels and permissible tolerances available shall guide the organisation to find the weak areas where actions may be prioritised.

1.13 QUALITY MANAGEMENT THROUGH CULTURAL CHANGES

Philip Crosby's approach to quality improvement is based on cultural change in an organisation towards total quality management. Quality management through cultural change defines quality improvements as a cultural change driven by management. It involves,

- Identifying areas in which quality can be improved depending upon process capability measurements and organisational priorities. An organisation must setup cross functional working groups (quality circles or quality improvement teams) and try to improve awareness about the customer needs, quality and process measurements. The organisation may not be able to improve in all areas at a time and prioritisation maybe essential depending upon some techniques such as Pareto analysis, Cost benefit analysis, etc. It must prioritise the improvements depending upon the resources available and efforts/investments required and the benefits derived from such improvements.

- Instituting teams representing different functions and areas for quality improvement can help in setting the change of culture. Improving quality of the processes of development, testing, managing, etc, is a team work led by management directives. A single person may not be able to institutionalise improvements across the Organisation, Improvements in processes automatically improve the product and customer satisfaction.

- Setting measurable goals in each area of an organisation can help in improving processes at all levels. Goals may at as stretched targets with respect to what is currently achieved by the organisation. Goals may be set with reference to customer expectations or something which may give competitive advantage to the organisation in the market.

- Giving recognition to achievers of quality goals will boost their morale and set a positive competition among the teams leading to organisational improvements. This can lead to dramatic improvements in all areas. Management must demonstrate commitment to quality improvement, and recognition of achievements is a step in this direction.

- Repeating quality improvement cycle continuously by stretching goals further for next phase of improvements is required to maintain and improve the status further. The organisation must evaluate the goals to be achieved in short term, long term, and the combination of both to realise organisational vision.

1.14 CONTINUAL (CONTINUOUS) IMPROVEMENT CYCLE

Plan, Do, Check, and Act (PDCA) Cycle:

Continual (Continuous) improvement cycle is based on systematic sequence of Plan Do-Check-Act activities representing a never-ending cycle of improvements. PDCA cycle was initially implemented in agriculture. It was implemented later in the electronic industry.TQM has made the PDCA cycle famous in all industries. PDCA improvement cycle can be thought of as a wheel of improvement continually (continuously) rolling up the problem-solving hill and achieving better and better results for the organisation in each iteration. Stages of continual (Continuous) improvement through PDCA cycle are,

Plan:

An organisation must plan for improvements on the basis of its vision and mission definition. Planning includes answering all questions like who, when, where, why, what, how, etc, about various activities and selling expectations. Expected results must be defined in quantitative terms and actions must be planned to achieve answers to these questions, Quality planning at unit level must be in sync with quality planning at organisation

level. Baseline studies are important for planning. Baseline studies define where one is standing and vision defines where one wishes to reach.

Do:

An organisation must work in the direction set by the plan devised in earlier phase for improvements. Plan is not everything but a roadmap. It sets the direction but execution is also important, Actual execution of a plan can determine whether the results as expected are achieved or not. Plan sets the tone while execution makes the plan work. 'Do' process need inputs like resources, hardware, software, training, etc. for execution of a plan.

Check:

An organisation must compare actual outcome of 'Do' stage with reference or expected results which are planned outcomes. It must be done periodically to assess whether the progress is in proper direction or not, und whether the plan is right or not, Expected and actual results must be in numerical terms, and compared at some periodicity as defined in the plan.

Act:

If any deviations (positive or negative) are observed in actual outcome with respect to planned results, the organisation may need to decide actions to correct the situation. The actions my include changing the plan, approach or expected outcome as the case may be. One may have to initiate corrective and or preventive actions as per the outcome of ‘Check’. When expected results and actuals match with given degree of variation, one may understand that the plan is going in the right direction. Running faster or slower than the plan will need action. Figure 1.4 shows diagrammatically PDCA cycle of continual improvement.

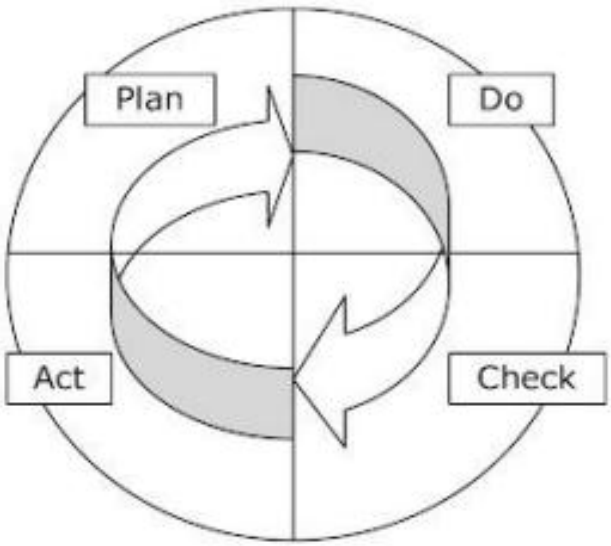


Fig. 1.4 Continual improvement cycle (PDCA cycle)

1.15 QUALITY IN DIFFERENT AREAS

Let us try to understand quality attributes of various products in different areas. Different domains need different quality factors. They may be derived from the customers/users of the domains. Here are few examples of some domains showing customer expectations in terms of quality for various products. These are generic expectations of customers in certain areas and may differ for some individual examples depending upon specific requirements. Definition of quality expectations will vary from instance to instance depending on the domain under consideration, type of product, type of customer, other competitive products, and their features. The table below lists different areas representing different domains and indicates some factors that might be considered related to quality in these areas. Below table shows some common expectations from customers.

Table 1.2 Products and expected attributes

Product/Service category	Expected attributes
Airlines Industry	On time arrival and departure, comfortable journey, low cost service, reliability and safety.
Health Care Industry	Correct diagnosis and treatment, minimum wait time, lower cost, safety and security.
Food Service Industry	Good product, good taste, fast delivery, good ambience, clean environment.
Consumer Products Industry	Properly made to suite individuals, defect free products, cost effective.
Military Services	Rapid deployment, decreased wages and cost, security.
Automotive Industry	Defect free product, less fuel consumption, more power, safe journey.
Communications Industry	Clear communications, faster access, cheaper service.

There are some common denominators in all these examples which may be considered as the quality factors. Although the terms used to explain each product in different domain areas vary to some extent, almost all areas can be explained in terms of few basic quality parameters. These are defined as the basic quality parameters by stalwarts of quality management.

- Cost of the product and value which the customer finds in it
- Service offered to the customers, in terms of support by the manufacturer
- Time required for the delivery of product
- Customer satisfaction derived from the attributes and functionalities of a product
- Number of defects in the product or frequency of failures faced by users

1.16 BENCHMARKING AND METRICS

Benchmarking is an important concept used in Quality Function Deployment (QFD). It is the concept of creating qualitative quantitative metrics or measurable variables, which can he used to assess product quality on several scales against a benchmark. Typical variables of benchmarking may include price of a product paid by customer, time required to acquire it, customer satisfaction, defects or failures, attributes and features of product, etc. **Metrics** are defined for collecting information about the product capabilities, process variability and outcome of the process in terms of attributes of products. Metric is a relative measurement of some parameters of a product which are related to the product and processes lived to make it. An organisation must develop consistent set of metrics derived from its strategic business plan and performance of benchmark partner.

1.17 PROBLEM SOLVING TECHNIQUES

Improving quality of products and services offered to customers requires methods and techniques of solving problems associated with development and processes used during their lifecycle. An organisation must use metrics approach of process improvement because it needs to make quantitative measurements. These measurements can be used for problem solving using quantitative techniques.

Problem solving can be accomplished by both qualitative and quantitative methods but problem definition becomes easier when we put them against some measures or comparators.

- Qualitative problem solving refers to understanding a problem solution using only qualitative indexes such as high, medium, low, etc. depending on whether something is improving or deteriorating from the present status and so forth. This is a typical scenario for low maturity organisations where the problems are much broader and can be classified in different bands very easily. For initial stages of improvement, qualitative problem solving is sufficient to get faster results. It saves time in defining and measuring data accurately and basic maturity can be achieved.

- Quantitative problem solving requires specification of exact measures in numerical terms such as the cost has increased 32.5% during the last quarter or the time required to produce one product unit is reduced by32 minutes. For highly matured organisations, quantitative analysis is required for further improvements as basic improvements are already done. It must follow the cycle of Define, Measure, Monitor, Control and Improve. Measurement of processes and products may need good measuring instruments with high level of accuracy and repeatability.

Given quantitative data, one can use statistical techniques to characterise a process. Quantitative methodologies make it possible to analyse and

visualise what is actually happening in a process. Process variations can be understood in a better way and actions can be initiated to reduce the variability.

1.18 PROBLEM SOLVING SOFTWARE TOOLS

While buying software for data management and statistical analysis, many organisations find it to be a big Investment in terms of money, resources, etc. One must answer the question 'Why should one use software tools to solve problems about quality?' There are some advantages and disadvantages associated with usage of such tools for problem solving,

Advantages of Using Software Tools for Analysis and Decision Making:

- Accuracy and speed of the tools is much higher compared to performing all transactions and calculations manually. Calculations can form the basis for miking decision, and hence should be as accurate as possible.
- Decision support offered by the tool is independent of personal skills and there is least variation from instance to instance. Tools can support in some fixed range depending upon its logic. Some tools can learn things and use them as required.
- Tools can implement theoretical means of assessing metrics about quality as defined by business law. There is no manual variation.
- Tools alleviate the hard work required to perform hand or calculator driven computations and give more accurate and faster results.
- Tools can be integrated with other systems to provide a systematic and highly integrated means of solving problems

Disadvantages of Using Computer Tools for Analysis and Decision Making:

- These programs and tools need training before they can be used. Training incurs cost as well as time. Some tools need specific trainings to understand them and use them.
- All softwares / hardwares are prone for defects and these tools are not exceptions to it. There can be some mistakes while building/using them. Sometimes these mistakes can affect the decisions drastically.
- Decision has to be taken by human being and not by the tool. Tools may define some options which may be used as guide. Some tools can take decision in the limited range.
- Tools may mean more cost and time to learn and implement. Every tool has a learning curve.

1.18.1 Tools:

Tools are an organisations analytical asset that assist in understanding a problem through data and try to indicate possible solutions. Quality tops are more specie tools which can be applied to solving problems faced by projects and functional teams while improving quality in organisations. Tools may be hardware/software and physical logical tools. We will learn more about quality tools in Chapter 16 on 'Qualitative and Quantitative Analysis'.

1.18.2 Techniques:

Techniques indicate more about a process used in measurement, analysis and decision-making during problem solving. Techniques are independent of tools but they drive tool usage. Techniques do not need tools for application while tools need techniques for then use. Same tool can be used for different purposes, if the need tools for application while tools need techniques for their use. Same tool can be used for different purposes, if the techniques are differed. Table 1.3 gives a difference between tools & techniques.

Table 1.3 Difference between tools and techniques

Tools	Techniques
Usage of tool is guided by the technique. Tool is of no use unless technique (to use it) is available.	Technique is independent of any tool.
Different techniques may use the same tool to achieve different results.	Same technique may use different tools to achieve the same result.
Tool improvement needs technological change.	Technique change can be effected through procedural change.
Contribution of tools in improvement is limited.	Contribution of techniques in improvement is important.

1.19 QUALITY TIPS

Try to define quality perspective for the organisation and set of products and projects executed by it.

- Define customer expectations rather than going for system requirement specifications.
- Assess the cost spent by an organisation under various heads of quality, Testers have to play a significant role in reducing cost of quality and improving profitability of the organisation.
- Understand and improve every aspect of an organisation through approaches, techniques and tools to enhance customer satisfaction and goodwill for the organisation. This can help the organisation to prosper in the long term.

1.20 SUMMARY

In this chapter, we have seen various definitions of quality as understood by different people and different stakeholders. It also covered the definitions by quality stalwarts and different international standards. Then,

we studied the basic components to produce quality, and the views of customers and producer on quality. As a tester, one must understand different gaps like user gap, and producer gap, and how to close them to achieve customer satisfaction.

We have described various cost components like manufacturing cost and cost of quality. Cost of quality concepts with its three components viz. preventive cost of quality, appraisal cost of quality and failure cost of quality are described along with the importance of cost of prevention, and how it affects cost of quality and improves profitability.

We have seen different approaches to continually (continuously) improving quality. We have covered 'TQM principles of quality management', and 'DMMCI principles of continual improvement' through quality planning, quality control and quality improvement. We have also discussed a theory of 'Cultural change principles of quality management'.

Finally, we have introduced the concept of problem solving through usage of tools and techniques. We have also briefly elucidated the concept of benchmarking.

1.21 EXERCISES

1) Explain 'quality' in terms of the generic expectations from any product.
2) Differentiate between continuous improvement and continual improvement.
3) Define the stakeholders for successful projects at micro level and for successful organisations at macro level.
4) Define 'quality' as viewed by different stakeholders of software development and usage.
5) Explain customer's view of quality.
6) Explain supplier's view of quality.
7) Define "User's gap' and 'Producer's gap' and explain how these gaps can be closed effectively.
8) Describe various definitions of quality as per international standards.
9) Describe definition of quality as per Dr Deming, Dr Juran and Philip Crosby.
10) Describe 'Total Quality Management' principles of continual improvement.
11) Describe cultural change requirement for quality improvement.
12) Differentiate between tools and techniques.

1.22 REFERENCES

- Software Testing andContinuous QualityImprovement by William E. Lewis, CRC Press, 3rdEdition, 2016.
- Software Testing:Principles, Techniquesand Tools by M. G. Limaye, Tata McGraw Hill, 2017.
- Foundations of SoftwareTesting by Dorothy Graham, Erikvan Veenendaal,Isabel Evans, RexBlack, CengageLearning, 3rd Edition.
- Software Testing: ACraftsman‟s Approach, Paul C. Jorgenson, CRC Press, 4th Edition, 2017.
- https://www.altexsoft.com/whitepapers/quality-assurance-quality-control-and-testing-the-basics-of-software-quality-management/
- https://www.softwaretestinghelp.com/software-quality-assurance

*****

2

SOFTWARE QUALITY

Unit Structure

2.0 OBJECTIVES

- This chapter focuses on software quality parameters.
- It provides a basic understanding that quality expectation for different products is different products is different and lays a foundation of quality management approach.

2.1 INTRODUCTION

In the previous chapter, we have discussed various definitions of quality and how they fit the perspective of different stakeholders. One of the definitions i.e.,

'**Conformance to explicitly stated and agreed functional and non-functional requirements**', may be termed as 'quality' for the software product offered to customers/final users from their perspective. Customers may or may not be the Final user's and sometimes, developers have to

understand requirements of final users in addition to the customer. If the customer is a distributor or retailer or facilitator who is paying for the product directly, then the final user's requirements may be interpreted by the customer to make a right product.

It is not feasible to put all requirements in requirement specifications. A large number of requirements remain undefined or implied as they may be basic requirements for the domain and the customer perceives them as basic things one must know. It gives importance to the documented and approved software and system requirement specifications on which quality of final deliverables can be decided. It shifts the responsibility of making a software specification document and getting it approved from customer to producer of a product. There are many constraints for achieving quality for software application being developed using such software requirement specifications.

2.2 CONSTRAINTS OF SOFTWARE PRODUCT QUALITY ASSESSMENT

Generally, requirement specifications are made by business analysts and system analysts. Testers may or may not have direct access to the customer and may get information through requirement statements, queries answered, etc. either from the customer or business system/analyst, etc. There are few limitations of product quality assessment in this scenario.

- Software is virtual in nature. Software products cannot be seen, touched or heard. Our normal senses and measuring instruments are not capable of measuring quality of software, which is possible in most of the other engineering products.

- There is a huge communication gap between users of software and developers/testers of the product. Generally, 5-8 agencies are involved between the users and developers/testers. If an average communication loss of 10% at each stage is considered, then huge distortion is expected from user's perspective of requirements and actual product.

- Software is a product which is unique in nature. Similarities between any two products are superficial ones. The finer requirements, designs foundation, architecture, actual code, etc. may be completely different for different products. Way of software design, coding, and reusability may differ significantly from product to product though requirements may look similar at a global level.

- All aspects of software cannot be tested fully as number of permutations and combinations for testing all possibilities tend to infinity. There are numerous possibilities and all of them may not be tried in the entire life cycle of a software product in testing or even in usage. Exhaustive testing is neither feasible nor justifiable with respect to cost.

- A software program executes in the same way every time when it is executing some instruction. An application with a problematic code executes wrongly every time it is executed. It makes a very small defect turn into a major one as the probability of defect occurrence is 100% when that part is executed.

Business analysts and system analysts must consider the following while capturing the requirements of a customer.

2.3 CUSTOMER IS A KING

The customer is the most important person in any process of developing a product and using it-software development is not an exception. All software life cycle processes such as development, testing, maintenance, etc. are governed by customer requirements, whether implied or expressed.

An organization must be dedicated to exceed customer satisfaction with the latter's consent. Exceeding customer satisfaction must not be received with surprise by the customer. He must be informed about anything that has been provided extra and must be in a position to accept it or reject it. Any surprise may be considered as defect.

A satisfied customer is an important achievement for an organization and is considered as an investment which may pay back in short as well as long term (in terms of references, goodwill, repeat order, etc.). Satisfied customers may give references to others and come back with repeat orders. Customer references are very important for developing new accounts. Organizations should try to implement some of the following measures to achieve customer satisfaction.

2.3.1 Factors Determining Success:

To be a successful organization, one must consider the following factors, entities, and their interactions with each other.

Internal Customer and Internal Supplier:

'Internal customer satisfaction' philosophy is guided by the principles of 'Total Quality Management'. When an organization is grouped into various functions/departments, then one function/department acts as a supplier or a customer of another function/department. If every function / department identifies its customers and suppliers and tries to fulfill their requirements, in turn, external requirements get satisfied.

External Customer and External Supplier:

External customers may be the final users, purchasers of software, etc. Software requirement specifications are prepared and documented by developing organizations to understand what customer/final user is looking for when he wishes to acquire a particular product. Customers are actually paying for getting their needs served and not for the implementation of the requirements as defined in the specifications

document. External suppliers are the entities who are external to the organization and who are supplying to the organization.

2.4 QUALITY AND PRODUCTIVITY RELATIONSHIP

Many people feel that better quality of a product can be achieved by more inspection or testing, reworking, scrapping, sorting, etc. More Inspection cycles mean finding more defects, fixing defects mean better quality (as it will expose maximum possible defects to be fixed by developers), and ultimately, the customer may get a better product. This directly means that there would be more rework/scrap and it will lead to more cost/price or less profit for such products as more effort and money is spend in improving quality. In such cases, time and effort required would be much higher.

In reality, quality improvements does not talk about product quality only but a process quality used for making such a product. If the processes of development and testing are good, a bad product will not be manufactured in the first place. It will reduce inspection, testing, rework, and cost/price. Thus quality must improve productivity by reducing wastage. It must target for doing 'first-time right'.

Improvement in Quality Directly Leads to Improved Productivity:

Improved quality does not mean more inspection, testing, sorting and rejection but improving the processes related to product development. All products are the outcome of processes, and good processes must be capable of producing good product at the first instance. This approach reduces frustration, rejection, rework, inspection, and improves quality and customer satisfaction. As this hidden factory producing scrap and wastage stops working, productivity and efficiency improves. Thus, quality products must give more profitability to the supplier and is a cheaper option for the customer.

The Hidden Factory Producing Scrap, Rework, Sorting, Repair, and customer Complaint is Closed:

Customer does not intend to pay for scrap, rework, sorting, etc. to get a good product. Engineering industry has faced this problem of unwillingness of customer pay even for first-time inspection/testing as they represent deficiencies in manufacturing processes. Problems in products can be linked to faulty development processes. Either the defects are not found in the product during process of development or testing, or fixing of the defects found in these processes introduces some more defects in the product offered.

Effective Way to improve Productivity is to Reduce Scrap and Rework by Improving Processes:

Productivity improvement means improving number of good parts produced per unit time and not the parts produced per unit time. It is not hard work but smart and intelligent work which can help an organization

in improving product quality, productivity and customer satisfaction by reducing rework, scrap, etc. It necessitates that an organization must incorporate and improve quality in the processes which lead to better product quality. As wastage reduce with improvements in quality in the processes which lead to better product quality. As wastage of resources reduce with improvement in quality, productivity and efficiency improves as a direct result by improvements in processes.

Quality Improvements Lead to Cost Reduction:

Quality improves productivity, efficiency and reduces scrap, rework, etc. Improvement in quality increases profit margins for producer by reducing cost of development, cost of quality and reduces sales price for customer. Thus quality implementation must reduce the cost and price without sacrificing profitability.

Employee Involvement in Quality Improvement:

Employee is the most important part of quality improvement program and crucial element for organizational performance improvement. Management leadership and employee contribution can make an organization quality conscious while lack of either of the two can create major problems in terms of customer dissatisfaction, loss of profitability, loss of goodwill, etc. Employee involvement in quality implementation is essential as the employee's facing problems in their work indicate the process problems. Management must include employees in quality improvement programs at all stages.

Proper Communication Between Management and Employee is Essential:

Communication is a major problem in today's environment. One of the communication hurdles is that the chances of face-to-face communication is by virtual appliances like emails, video conferencing, etc. Where it is difficult to judge the person through body language. Either there is no communication or there is excessive communication leading to a problematic situation. There are huge losses in communication and distortions leading to miscommunication and wrong interpretation. The reasons for 'producer's gap' and 'user's gap' are mainly attributed to communication problems. Different words and terms used during the message, tone, type of message, speaking skills, listening skills, etc. contribute to quality of communication.

Employees Participate and Contribute in Improvement Process:

In quality improvement program and implementation, employees perform an important part of identification of problems related to processes and giving suggestions for eliminating them. They are the people doing actual work and know what is wrong and what can be improved in those processes to eliminate problems. They must be closely associated with the organization's goal of achieving customer satisfaction, profitability, name and fame in market, etc. Every employee needs to play a part in

implementation of 'Total Quality Management' in respective areas of working. This can improve the processes by reducing any waste.

Employee Share Responsibility for Innovation and Quality Improvement:

Management provides support, guidance, leadership, etc. and employees contribute their part to convert organization into performing teams. Everyday work can be improved significantly by establishing small teams for improvement which contribute to innovations. An organization must not wait for inventions to happen for getting better products but perform small improvements in the processes every day to achieve big targets in the long range. The theory of smart work in place of hard work helps employees to identify any type of waste in the process of development and eliminate it.

TABLE 2.1 Difference between invention and innovation:

Invention	Innovation
• Inventions may be accidental in nature. They are generally unplanned. • Invention may or may not be acceptable to people doing the work immediately. Inventions are done by scientist and implementation and acceptance by people can be cumbersome as general level of acceptance is very low. • Inventions may not be directly applied to everyday work. It may need heavy customization to make it suitable for normal usage. • Breakthrough changes are possible due to inventions. • Invention may lead to major changes in technology, way of doing work, etc. • Invention may lead to scraping of old technologies, old skills and sometimes, it meets with heavy resistance.	• Innovation is a planned activity leading to changes. • Innovation is done by people in a team, possibly cross-functional teams, involved in doing a work. There is higher acceptability by people as they are involved in it. • Innovations can be applied to every day work easily. The existing methods and processes are improved to eliminate waste. • Changes in small steps are possible by innovation. • Innovation generally leads to administrative improvements, whereas technological or breakthrough improvements are not possible. • Innovation may lead to rearrangement of things but there may not be a fundamental change. It generally works on elimination of waste

Table 2.1 Gives a difference between invention & innovation.

Many organisations have a separate 'Research and Development' function responsible for doing inventions. These functions are dedicated to make breakthrough changes by developing new technologies and techniques for accomplishing work. They are supposed to derive major changes in the approaches of development, implementation, testing, etc. 'Six sigma' improvements also talk about breakthrough improvements where processes may be redesigned/redefined. It may add new processes and eliminate old processes, if they are found to be problematic. But, an organisation should also create an environment which helps in innovation or rearrangement of the tasks to make small improvements everyday. Continuously identifying any type of waste in day-to-day activities and removing all nonessential things can refine the processes.

2.5 REQUIREMENTS OF A PRODUCT

Everything done in software development, testing and maintenance is driven by the requirements of a product to satisfy customer needs. There are basic requirements for building software, which will help customers conduct their businesses in a better way. Every product offered to a customer is intended to satisfy some requirements or needs of the customer. Requirements may be put in different categories. Some of these are:

Stated/Implied Requirements:

Some requirements are specifically documented in software requirement specifications while few others are implied ones. When we build software, there are functional and non-functional requirements specified by a customer and/or business/system analyst. It is also intended not to violate some generally accepted requirements such as "No spelling mistakes in user interfaces, *Captions on the control must be readable', etc. These types of requirements may not be documented as a part of requirement statement formally but generally considered as requirement, and are expected in a product. As a part of development team/test team, one must understand stated as well as intended or implied requirements from the users. It may be a responsibility of a developing organisation to convert as many implied requirements as possible into stated requirements. Though impossible, the target may be 100%.

General/Specific Requirements:

Some requirements are generic in nature, which are generally accepted for a type of product and for a group of users while some others are very specific for the product under development. Addition of two numbers should be correct is a generic requirement while the accuracy of 8 digits after decimal and rounding may be a very specific requirement for the application underdevelopment. Usability is a generic requirement in software for the intended user while authentication and messaging to users may be driven by specific requirements of an application. Many times, the generic requirements are considered as implied ones and those may not be mentioned in requirement specifications while specific requirements are stated ones as those are present in requirement specifications.

Present/Future Requirements:

Present requirements are essential when an application is used in present circumstances while future requirements are for future needs which may be required after sometime span. For projects, present as well as future requirements may be specifically defined by a customer or business/system analyst. A product development organisation may have to do further research to identity or extrapolate future needs of users. For banking software, today's requirements may be 5000 saving accounts, and the application may be running in client-server environment. But a future need may be 50 lakhs saving accounts and the application may be running as a web application. *How much future must be guided by the customer's vision as this may influence product cost. Definition of future has direct relationship with usable life of an application. Some people may use a software for 3 years while some other may be planning to use it for 30 years. The future requirements may change as per the expected life span of the software. On the basis of priority of implementation from user's perspective, requirements may be categorised indifferent ways as follows:

'Must' and 'Must not' Requirements or Primary Requirements:

‘Must’ requirements are primary requirements for which the customer is going to pay for while acquiring a product. These are essential requirements and the value of the product is defined on the basis of the accomplishment of ‘must’ requirements. Generally these requirements have the highest priority in implementation. Not meeting these requirements can cause customer dissatisfaction and rejection of a product. These requirements may be denoted by priority ‘PI’ indicating the highest priority. It also covers "Must not' requirements which must be absent in the product.

'Should be' and 'Should not be' Requirements or Secondary Requirements:

‘Should be’ requirements are the requirements which may be appreciated by the customer it they are present/absent and may add some value to the product. Customer may pay little bit extra for the satisfaction of these requirements but price of the product is not governed by them. Generally, these requirements are lower priority requirements than 'Must' requirements. These requirements may give the customer delight, if present and little disappointment, if absent. These requirements may be denoted by priority ‘P2’. It also covers ‘Should not’ requirements.

'Could be' and 'Could not be' Requirements or Tertiary Requirements:

‘Could be' requirements are requirements which may add a competitive advantage to the product but may not add much value in terms of price paid by a customer. If two products have everything same, then could be requirements may help in better appreciation of a product by the users. These are the lowest priority requirements. It also covers ‘Could not’ requirements. These requirements give a product identity in the market. These requirements may be denoted by priority ‘P3’. While talking about

a product in view of a bigger market with large number of generic users, it may be very difficult to categorise the requirements as mentioned above. This is because "must' requirement for one customer may be *could be' requirement for somebody else. In such cases, an organisation may have to target the customer segment and define the priorities of the requirements accordingly. Customer must have the final authority to define the category of requirement.

2.6 ORGANISATION CULTURE

An organisation has a culture based on its philosophy for existence, management perception and employee involvement in defining future. Quality improvement programs are based on the ability of the organisation to bring about a change in culture. Philip Crosby has prescribed quality improvement for cultural change. Quality culture of an organisation is an understanding of the organisations virtue about its people, customer, suppliers and all stakeholders, *Q' organisations are more quality conscious organisations, while *q' organisations are less quality conscious organisations. The difference between 'Q' organisations and 'q' organisation is enumerated as follows.

Table 2.2Difference between 'Q' organisation and 'q' organization:

Quality culture is 'Q'	Quality culture is not 'q'
• These organisations believe in listening to customers and determining their requirements. • These organisations concentrate on identifying cost of quality and focusing on it to reduce cost of failure which would reduce overall cost and price. • Doing things right for the first time and every time is the motto of success. • They concentrate on continuous/continual process improvement to eliminate waste and get better output. • These organisations believe in taking ownership of processes and defects at all levels. • They demonstrate leadership and commitment to quality and customer satisfaction.	• These organisations assume that they know customer and requirements. • These organisations overlook cost of poor quality and hidden factory effect. They believe in more testing to improve product quality. • Doing things again and again to make them right is their way of working. Inspection, rework, scrap, etc. are essential. • They work on the basis of finding and fixing the problem as and when it is found. Onetime fix for each problem after it occurs. • These organisations try to assign responsibility of defects to someone else. • They believe in assigning responsibility for quality to others.

2.6.1 Shift In Focus From 'q'to'Q':

As the organisation grows from ‘q’ to ‘Q’, there is a cultural change in attitude of the management and employees towards quality and customer. In initial stages, at the level of higher side of *q', a product is subjected to heavy inspection, rework, sorting, scrapping, etc., to ensure that no defects are present in final deliverable to the customer while the final stages of Q' organisation concentrate on defect prevention through process improvements. It targets for first-time right. Figure 2.1 shows an improvement process where focus of quality changes gradually.

Fig. 2.1 Shift in focus from Quality control to Quality management

Quality Control Approach (Finding and Fixing Defects):

Quality control approach is the oldest approach in engineering when a product was subjected to rigorous inspection for finding and fixing defects to improve it. Organisations at the higher end of *q' believe in finding and fixing defects to the extent possible as the way to improve quality of product before delivering it to customer and achieving customer satisfaction. Basically, works on correction attitude involving defect fixing, scrap, rework, segregation, etc. It works on the philosophy that a product is good unless a defect is found in it. There are huge testing teams, large investment in appraisal cost and defect fixing costs followed by retesting and regression testing.

Quality Assurance Approach (Creation of Process Framework):

Quality assurance is the next stage of improvement from quality control where the focus shifts from testing and fixing the defects to first-time right. An Organisation does investment in defining processes, policies, methods for handling various functions so that it can incorporate a process approach for doing various things, It becomes a learning organisation as it shifts its approach from 'quality control' to 'quality assurance'. The management approach shifts from corrections to corrective actions through root cause analysis. There are some actions on the basis of metrics program instituted by the organisation. It also starts working on preventive actions to some extent to avoid potential defects, Defects are considered as failures of processes and not of people, and actions are initiated to optimise the processes.

Quality Management Approach:

There are three kinds of system in the universe, viz. completely closed systems, completely open systems and systems with semi permeable boundaries, completely closed systems represent that nothing can enter inside the system and nothing can go out of the system. On the other hand, open system represents a direct influence of universe on system and vice-a-versa. Completely closed systems or completely open systems do not exist in reality. Systems with semi permeable boundaries are the realities, which allow the system to get impacted from external changes and also have some effect on external environment. Anything coming from outside may have an impact on the organisation but the level of impact may be controlled by taking some actions. Similarly, anything going out can also affect the universe but impact is controlled. Organisations try to assess the impact of the changes on the system and try to adapt to the changes in the environment to get the benefits. They are highly matured when they implement Quality Management as a management approach. There are virtually no defects in processes as they are optimised continuously, and products are delivered to customers without any deficiency. The organisation starts working on defect prevention mechanism and continuous improvement plans. The organisation defines methods, processes and techniques for future technologies and training programs for process improvements. Management includes planning, organising, staffing, directing, coordinating, and controlling to get the desired output. It also involves mentoring, coaching, and guiding people to do better work to achieve organisational objectives.

2.7 CHARACTERISTICS OF SOFTWARE

There are many products available in the market which are intended to satisfy same or similar demands. There is a vast difference between software products and other products due to their nature. Software cannot be sensed by common methods of inspection or testing, as it is virtual in nature. The product is in the form of executable which cannot be checked by any natural method available to mankind like touch, smell, hearing, taste, etc. It cannot be measured by some measuring instruments commonly available like weighing balance, scales, etc. It needs testing in real environment but nobody can do exhaustive testing by trying all permutations and combinations. There are different kinds of software products and their performance, capabilities, etc. vary considerably from each other. There are no same products though there may be several similar ones or satisfying similar needs. Every product is different in characteristics, performance, etc. Software is always unique in nature. Every condition defined by the software program gets executed in the same way every time when it gets executed. But the number of conditions, and algorithm combinations may be very large tending to infinity and testing of all permutations/combinations is practically impossible.

2.8 SOFTWARE DEVELOPMENT PROCESS

Software development process defines how the software is being built. Some people also refer to SDIC as system development life cycle with a view that system is made of several components and software is one of these components. There are various approaches to build software. Every approach has some positive and some negative points. Let us talk about few basic approaches of developing software from requirements. It is also possible that different people may call the same or similar approach by different

- Waterfall development approach/model
- Iterative development approach/model
- Incremental development approach model
- Spiral development approach/model
- Prototyping development approach/model
- Rapid application development approach/model
- Agile development approach/model.

2.8.1 Waterfall Development Approach/Model:

Waterfall model is the simplest software development model and is used extensively in development process study. There are many offshoots of waterfall model such as modified waterfall model, iterative waterfall model, etc. Though it is highly desirable to use waterfall model, it may not be always feasible to work with it. Still, waterfall model remains as a primary focus for study purpose. It is also termed as classical view of software development as it remains the basis or foundation of any development activity. Most of the other models of development are based upon the basic waterfall model as it represents a logical way of doing things. Typical waterfall model is shown in Fig. 2.2.

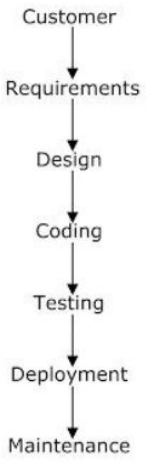


Fig. 2.2 Waterfall model

Arrows in the waterfall model are unidirectional. It assumes that the developers shall get all requirements from a customer in a single go. The requirements are converted into high level as well as low level designs. Designs are implemented through coding. Code is integrated and executables are created. Executables are tested as per test plan. The final output in the form of an executable is deployed at customer premises. Future activities are handled through maintenance. If followed in reality, waterfall model is the shortest route model which can give highest efficiency, and productivity. Waterfall models are used extensively in fixed price/fixed schedule projects where estimation is based on initial requirements. As the requirement changes, estimation is also revised.

Limitations of Waterfall Cycle:

There is no feedback loopavailable in waterfall model. It is assumed that requirements are stableand no problem is encountered during entire development life cycle.Also, no rework is involved in waterfall model.

2.8.2 Iterative Development approach/Model:

Iterative development process is more practical than the water fall model. It does not assume that the customer gives all requirements in one go and there is complete stability of requirements. It assumes that changes may come from any phase of development to any previous phase and there are multiple permutations and combinations of changes. Changes may have a cascading effect where one change may initiate a chain reaction of changes. Figure 2.3 shows a feedback loop which is the fundamental difference between waterfall model and iterative development model.

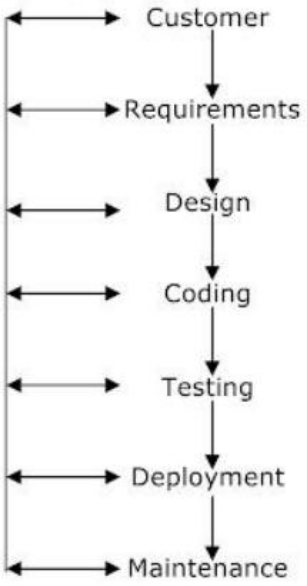


Fig. 2.3 Iterative development model

Limitations of Iterative Development:

Iterative development consists of many cycles of waterfall model. It gives problems in fixed price projects for estimation. An-other problem faced by

iterative development is that the product architecture and design becomes fragile due to many iterative changes.

2.8.3 Incremental Development approach/Model:

Incremental development models are used in developing huge systems. These systems are made of several subsystems which in themselves are individual systems. Thus, incremental systems may be considered as a collection of several subsystems. An individual subsystem May be developed by following waterfall methodology and iterative development. These subsystems may be connected to each other externally, either directly or indirectly. A directly interconnected system allows the subsystems to talk with each other while indirectly interconnected system has some interconnecting application between two subsystems. Direct connectivity makes a system more robust but flexibility can be a major issue. The incremental model gives flexibility to a customer. One system may be created and the customer may start using it. The customer can learn the lessons and use them while second part of the system is developed. Once those are integrated, third part may be developed and so on. The customer does not have to give all requirements at the start of development phase.

The model in Fig. 2.4 shows a common communication system and incremental subsystems where different subsystems communicate through a common communication system.

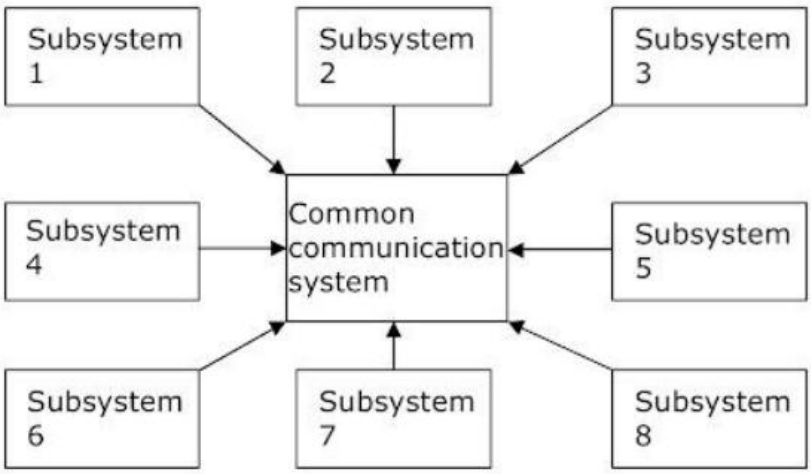


Fig. 2.4 Incremental model

Limitations of Incremental Development:

Incremental models with multivendor product integration are a major challenge as parameter passing between different systems may be difficult. Incremental models help in integration of big systems at the cost of loss of flexibility. When a system is incremented with newsub systems, it changes the architecture of that system. Increment in the system is followed by heavy regression testing to find that when multiple systems come together, can they work individually as well as collectively.

2.8.4 Spiral Development Approach/Model:

Spiral development process assumes that customer requirements are obtained in multiple iterations, and development also works in iterations. Many big software systems are built by spiral models of ever-increasing size. First some functionalities are added, then product is created and released to customer. After getting the benefits of first iteration of implementation, the customer may add another chunk of requirements to the existing one. Further addition of requirements increase the size of the software spirally. Sometimes, an individual part developed in stages represents a complete system, and it may communicate with the next developed system through some interlaces. In many ERP, initial development concentrated around material management part which later increased spirally to other parts such as purchasing, manufacturing, sales, warehousing, cash control, etc. Many banking software's also followed a similar route. Figure 2.5 shows a spiral development model.

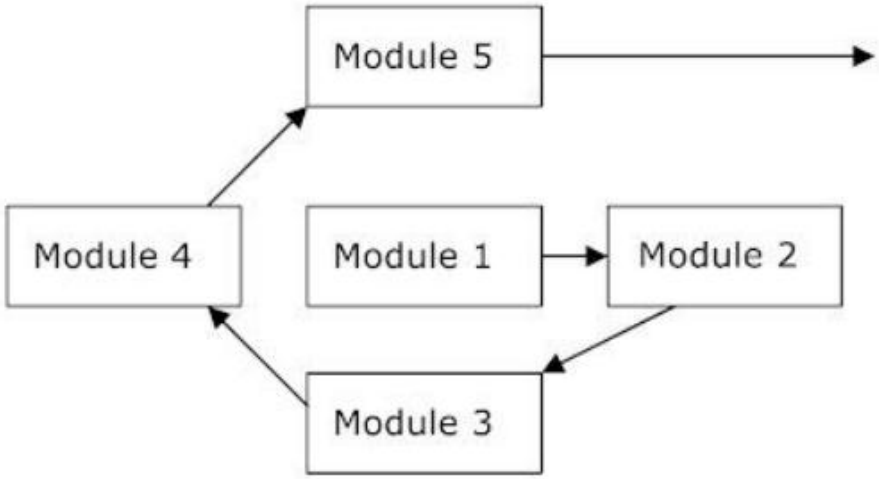


Fig. 2.5 Spiral development model

- **Limitations of Spiral Development Spiral** models represent requirement elicitation as the software is being developed. Sometimes, it may lead to refactoring and changes in approach where initial structures become n on-usable. Spiral development also needs huge regression testing cycles to find whether additions in the given system have affected overall system working or not.

2.8.5 Prototype Development Approach/Model:

Prototype development approach represents top to bottom reverse integration approach. Major problem of software development is procuring and understanding the customer requirements for the product. Prototyping is one of the solutions to help in this problem. In prototyping, initially a prototype of the system is created- this is similar to cardboard model of a building. It helps the customer to understand what they can expect from the given set of requirements. It also helps the development team to understand the possible application's look and feel. Once the

elicitation is done, the logic is built behind it to implement the elicited requirements.

Limitations of Prototype Development:

Though, one may get a feel of the system by looking at the prototype, one must understand that it is not the actual system but a model. The customer may get the feeling that the system is already ready and may pressurise development team to deliver it immediately. (Applications not having much graphical user interfaces are difficult to model.)

2.8.6 Rapid Application Development Approach/Model:

Rapid application development is not a rapid way of developing software as one may interpret from the name. It is one way to create usable software at a fast speed, and still give an opportunity to the user to understand the development and application being created. It is a miniature form of spiral development. Development team may get very less number of requirements (let us say 5/6). They create a design, code it, test it and release it to customer. Once customer gets the delivery, he may have a better understanding of his expectations and development process by looking at the product delivered. He may add another chunk of requirements and entire development cycle is followed. Thus, each iteration will give better understanding about a product being developed and may help in refining the requirements.

Limitations of Rapid Application Development:

Change in approach and refactoring are the major constraints in rapid application development. It also involves huge cycles of retesting and regression testing. Efforts of integration are huge.

2.8.7 Agile Development Approach/Model:

Agile development methodologies are becoming popular due to their dynamic nature and easy adaptability to the situation. One of the surveys indicated that in case of waterfall model, many functionalities are added in requirement statement with a fear that changes in scope would not be appreciated by the development team. Some surveys show that many of the functionalities (about ¾th) developed using waterfall or iterative model are never used by the users. Agile gives complete freedom to the user to add requirements at any stage of development, and development team has to accept these changes. Agile methodologies work on small chunk of work in each iteration and release working software at the end of iteration. The main thrust of Agile methodologies is complete adaptability to user environment and continuous integration of a product. It also gives importance to delivering working software rather than achieving requirements defined in requirement specifications. Agile represents a family of development methodologies and there are many methodologies under its umbrella. Some of them are as listed below.

- Scrum

- Extreme Programming
- Feature Driven Development
- Test Driven Development

Agile works on the following principles,

- Individuals and interactions are more important than formal sign-offs for requirements, designs, etc. It concentrates more on "Fitness for use' and what the customer needs are.
- Working software is the outcome of each milestone rather than concentrating on deliverables as defined in the project plans. Success of software product is that it is working at each stage.
- Customer collaboration is required to get usable software rather than signing various documents tor approvals. Requirement clarifications, requirement elicitation, and prototyping need customer involvement.
- Responding to changes required by the customer at any moment. There may be many changes suggested by the customer as he has better knowledge about what his business needs are.

2.8.8 Maintenance Development Approach/Model:

Major cost of the software is in its maintenance phase. Every product including software has many defects which may create problems to its users in the long term. Every technology has a life span. New technologies may offer better services and options, and may replace existing technologies, every now and then, technological updations are required for the software as well as system to perform better and in the most cost effective way. New functionalities may be required due to changing business needs. Maintenance activities of software may be put under 4 different groups namely,

- **Bug fixing** where the defects present in the given software are fixed. This may involve retesting and regression testing. During bug fixing, analysis of bug is an important consideration. There is always a possibility that while fixing a bug, new bugs may have been added in the product.
- **Enhancement** where new functionalities are added in the existing software. These functionalities maybe required due to changes in the way business is done. Some functionalities may be introduced due to changes in user requirements.
- **Porting** where software is taken from older technologies to newer technologies. In porting, one is expected to port the functionalities and not the code. Whatever functionalities are available in the old technologies; all those are expected to be present in the new technology.

- **Reengineering** where there is some change in the logic or algorithm used due to changes in business environment.

2.9 LIFE AFFECTING PRODUCTS

Similar to software development methodologies, software products have some peculiarities defined as criticalities of software. Criticality of the software defines; how much important it is to a user/customer. There are various schemes of grouping the products on the basis of criticality to the users. Few of them are listed below,

2.9.1 Life Affecting Products:

Products which directly/indirectly affect human life are considered as the most critical products in the world from user's perspective. Such products generally come under the purview of regulatory and safety requirements, in addition to normal customer requirements. The quality requirements are more stringent, and testing is very critical for such products as failures may result into loss of life or disablement of a user. This type of product may be further grouped into 5 different categories.

- Any product failure resulting into death of a person. These will be the most critical products as they can affect human life directly.
- Any product failure which may cause permanent disablement to a patient. These are second-level criticality products.
- Any product failure which may cause temporary disablement to a patient
- Any product failure which may cause minor injury and does not result into anything as defined above
- Other products which do not affect health or safety directly

Such software needs huge testing to try each and every conceivable fault in the product. It talks about very high level of confidence that application will not fail under normal and abnormal situations.

2.9.2 Product Affecting Huge Sum of Money:

A product which has direct relationship with loss of huge sum of money is second in the list of criticality of the product. Such products may need large testing efforts and have many regulatory as well as statutory requirements, Commerce and Business software may be put in this category. Security, confidentiality, and accuracy are some of the important quality factors for such products.

These products also need very high confidence level and huge testing will represent the criticality. They need testing lesser than the products which directly/indirectly affect human life.

2.9.3 Products Which Can Be Tested Only By Simulators:

Products which cannot be tested in real-life scenario but need simulated environment for testing are third in the ranking of criticality. In this case, real life scenario is either impossible to create or may not be economically viable. Products used in aeronautics, space research, etc. may be put in this category.

Such products also need huge testing, although lesser than the earlier two types.

2.9.4 Other Products:

All other products which cannot be categorised in any of the above schemes may be put in this category.

Unfortunately, "criticality' is not very easy to define. Let us consider an example of auto piloting software where we have three combinations of criticality together. It does affect the life of passengers traveling, cost of an aeroplane is huge and it cannot be tested in real environment. Thus, all the three criticalities coming together make a product most critical.

2.10 SOME OTHER SCHEMES OF CRITICALITY DEFINITIONS

There may be several other ways of classifying criticality of a product. It has direct relationship with business dependability and the extent of loss to the user organisation person in case of failure.

2.10.1 From User's Perspective:

This classification mainly discusses dependency of a business on a system. The criticality may range from complete dependency to no/minimal dependency on the system.

- Product's failure which disrupts the entire business can be very critical from business point of view. There is no fall back arrangement available or possible in case of failure of a product which is completely dependent on the system.

- Product's failure which affects business partially as there may be some fall back arrangements is a type of criticality. Business may be affected temporarily, affecting profitability or service level but can be restored with some efforts.

- Product's failure which does not affect business at all is one of the options. If it fails, one may have another method to achieve the same result. Rearrangement may not have significant distortion of business process.

2.10.2 Another Way of Defining User's Perspective:

This classification considers the environment in which the product is operating. It may range from very complex user environment to very easy user environment.

- Products where user environment is very complex such as aeronautics, space research, etc., may be considered as very critical. Any failure of product can result into major problems as the environment where these products are working is not an easy environment. As the environment is very complex, system is already under stress, and failure of a product may add to the situation.

- Products where user environment is comparatively less complex (such as banks) may represent the second stage of complexity. Huge calculations may be affected, if the system collapses but there maybe workaround available. People may find it inconvenient, but still the operations can be performed with some tolerable level of problems. For example, banks were running for so many years without computers and if centralised system fails, they may be able to withstand the pressure to some extent.

- Products where user environment is very simple, and product failure may not add to the consequences represents the lowest level of complexity. If there is any failure, it can be restored quite fast or some other arrangements can be used, and work can be continued.

2.10.3 Criticality from Developer's Perspective:

This classification defines the complexity of the system on the basis of development capabilities required. It may range from very complex systems to very simple systems.

- Form based software where user inputs are taken and stored in some database. As and when required, those inputs are manipulated and shown to a user on screen or in form of a report. There is not much manipulation of data and no heavy calculations/algorithms are involved.

- Algorithm based software, where huge calculations are involved and decisions are taken or prompted by the system on the basis of outcome of these calculations. Due to usage of different mathematical models, the system becomes complex and designing, developing and testing all combinations become problematic.

- Artificial intelligent systems which learn things and use them as per circumstances are very complex systems. An important consideration is that the 'learnings' acquired by the system must be stored and used when required. This makes the system very complicated.

There may be a possibility of combination of criticalities to various extends for different products at a time. A software used in aviation can

affect human life, and also huge money at a time. It may not be feasible to test it in real-life scenario. It may involve some extent of artificial intelligence where system is expected to learn and use those "learnings'. Thus, the combination increases severity of failure further.

2.11 PROBLEMATIC AREAS OF SOFTWARE DEVELOPMENT LIFECYCLE

Let us discuss some problematic areas of software development life cycle.

2.11.1 Problems with Requirement Phase:

Requirement gathering and elicitation is the most important phase in software development life cycle, Many surveys indicate that the requirement phase introduces maximum defects in the product. Problems associated with requirement gathering are,

Requirements Are Not Easily Communicated Communication is a major problem in requirement statement creation, software development and implementation. Communication of requirements from customer to development team is marked by problems of listening to customer, understanding business domain, and usage of language including domain specific terms and terminologies. The types of requirements are,

Technical Requirements:

Technical requirements talk about platform, language, database, operating system, etc. required for the application to work, Many times, the customer may not understand the benefits of selectinga specific technology over the other options and problems of using these technically specified configurations.

Selection of technology may be done as directed by the development team or as a fashion. Development organisation is mainly responsible for definition of technical requirements for software product under development on the basis of product usage. (Technical requirements also cover this type of system, whether a standalone or client server or web application etc, tiers present in the system, processing options such as online, batch processing, etc). It also talks about configuration of machines, routers, printers, operating systems, databases, etc.

Economical Requirements:

Economies of software system is dependent on its technical and system requirements. The technical as well as system requirements may be governed by the money that the customer is ready to put in software development, implementation and use. It is governed by cost-benefit analysis.

These requirements are defined by development team along with the customer. The customer must understand the benefits and problems associated with different approaches, and select the approach on the basis

of some decision-analysis process. The consequences of any specific selection may be a responsibility of the customer but development organisations must share their knowledge and experience to help and support the customer in making such a selection.

Legal Requirements:

There are many statutory and regulatory requirements for software product usage. For any software application, there may be some rules and regulations by government, regulatory bodies, etc. applicable to the business. There may be some rules which keep on changing as per decisions made by government, regulatory authorities, and statutory authorities from time to time. There may be numerous business rules which are defined by customers or users for doing business. Development team must understand the rules and regulations applicable for a particular product and business.

Operational Requirements:

Mostly operational requirements are defined by customers or users on the basis of business needs. These may be functional as well as non-functional requirements. They tell the development team, what the intended software must do/must not do when used by the normal user. Operational requirements are derived from the business requirements. This may include non-functional requirements like security, performance, user interface, etc.

System Requirements:

System requirements including physical/logical security requirements are defined by a customer with the help of a development team. These include requirements for hardware, machine configurations, types of backups, restoration, physical access control, etc. These requirements are defined by customer's management and it affects economies of the system. There may be some specific security requirements such as strong password, encryption, and privilege definitions, which are also declared by the customer.

Requirements Change Very Frequently:

Requirements are very dynamic in nature. There are many complaints by development teams that requirement change is very frequent. Many times, development teams get confused because customer requirements change continuously. **'Customer does not know what he wants'** is a very common complaint made by development teams. As the product is being built and shown to customer, lot of new ideas are suggested. Some ideas may have significant effect on cost, schedule, and effort while some other may change the architecture, basic approach, and design of software. The time gap between requirement definition and actual product delivery also plays a major role in changing requirements. Top-down approach, rapid application development, and joint application development are some of

the techniques used to develop applications by accommodating changes suggested by customers.

2.11.2 Generally A Unique Product Is Developed Each time:

No two things in the world are same, though they might appear to be similar. In case of software no two applications are same. Even in case of simple porting (desktop to client-server to web application), software application changes significantly. The same implementation done by two different developers may differ from each other. Even the same program written by the same developer at two different instances may not match exactly. Thus a software produced may be unique for that instance. One more fact about software product maintenance is that designers find it difficult to understand original design or approach and developers find it difficult to read the code written earlier.

2.11.3 Intangible Nature of Product, Intellectual Approach:

Throughout Development:

Software products cannot be felt by normal senses. Its existence can be felt only by disc space it occupies. There are multiple options or approaches (for example, in architecture or design) possible for implementation of the same set of requirements. Some may feel that one approach is better than the other for different reasons. The capabilities of individuals and organisations vary significantly in design and development, and each may have a good justification why a certain approach is selected with respect to other.

2.11.4 Inspection Can Be Exhaustive/Impossible:

While defining exhaustive inspection, one may tend to include infinite permutations and combinations of testing. Testing of complete software product is practically impossible. It may need huge money and longtime to test all possibilities, and still one may not be sure that everything is covered in testing. Testing uses a sampling theory to find the defects in the product and processes used. Testing tries to find out the lacunae in development methodology and processes used.

2.11.5 Effect of Bad Quality Is Not Known Immediately:

Any level of exhaustive testing is not capable of testing each and every algorithm, branch, condition and combination thoroughly. There are some areas which remain untested even after the application is used overextended periods. Any problem in such areas may be discovered only when the particular situation arises. The effect of this kind of problem and situation during usage may not be known beforehand while deploying the software in use.

2.11.6 Quality is Inbuilt in Product:

Quality of a software product cannot be improved by testing it again and again and finding and fixing the defects. It needs to be built in the product

while development using good processes and methods. Any amount of testing cannot certify that a product is defect-free. Good processes and procedures can make good software. No software can be considered as defect-free even if no defect is found in the test iteration defined for it. We can only say that no defect has been discovered till that point of time using those many test cases.

2.11.7 Quality Objectives Vary From Product To Production:

Customer to Customer:

Quality objectives define the expectations of customer/user and the acceptance level of various parameters which must be present in a given product for accepting/using it. Quality objectives are product dependent, time dependent and are mainly driven by customers or final users. There may be a possibility of trade-off between these factors. Some people define them as test objectives as they define the priority of testing. Ina small computer game for kids, cost may be more important than accuracy. On the contrary, applications developed for aeronautics need to be more accurate while cost factor may not be that important. Quality objectives are defined on the basis of factors of quality which are 'must', 'should be', and 'could be' for the application. Degree of importance changes from product to product, customer to customer, and situation tosituation. Some of the quality factors are listed below.

Compliance:

Every system is designed in accordance with organisational and user policies, procedures and standards. If software meets these standards, it is said to be complying with the specifications. In addition to customer defined standards, there may be few standards for different domains like aviation, medical, and automotive. Software must follow these standards when it is used by particular type of people or for a particular domain. Some regulations and laws may be imposed by the regulatory and statutory bodies.

Generally, these requirements are categorised as legal requirements. These requirements may be put in non-functional requirements.

Correctness:

Data entered, processed and results obtained must be accurate as per requirements of customers and/or users. Definition of correctness may change from customer to customer, application to application, and time to time. Generally, accuracy refers to mathematical accuracy, but it is not a rule. For a shopkeeper, accuracy of 0.01 may be sufficient as nothing below 1 paisa is calculated while a scientist may need an accuracy of 256 digits after decimal point as rounding off errors can cause a major problem in research work.

Ease of Use:

Efforts required to learn, operate, prepare input for and interpret output from the system define ease of use for an application. The normal users

who are expected to use the software must be comfortable while using it. Ease of use reduces training cost for the new users dramatically. If a software application can be learned without any external help, such software is considered as the best from this perspective. If there is a requirement of training or hand holding before the software can be used, people may find it inconvenient. Ease of use is a very important factor when large number of users are expected (for example, emailing, software and mobile phones), and providing them training is a difficult task.

Maintainability:

Efforts required to locate and fix errors in an operational system must be as less as possible to improve its ability for maintenance. There may be some possibilities of enhancements and re-engineering where good maintainable software has least cost associated with such activities. Software may need maintenance activity some time or the other to improve its current level of working. Ability of software to facilitate maintenance is termed as maintainability. Good documentation, well commented code, and requirement traceability matrix improve maintainability of a product.

Portability:

Efforts required in transferring software from one hardware configuration and/or software system environment to another environment defines portability of a system. It may be essential to install same software in different environments and configurations as per business needs. If the efforts required are less, then the system may be considered as highly portable. On the other hand, if the system cannot be put in a different environment, it may limit the market.

Coupling:

Coupling talks about the efforts required in interconnecting components within an application and interconnection of system as a whole with all other applications in a production environment. Software may have to communicate with operating system, database, and other applications when a common user is working with it. Good coupling ensures better use of environment and good communication, while bad coupling creates limitation on software usage.

Performance:

Amount of resources required to perform stated functions define the performance of a system. For better and faster performance requirements, more and more system resources and/or optimised design may be required. Better performance improves customer satisfaction through better experience for users. Performance attribute may cover performance, stress and volume. Details about these will be discussed in the latter chapters of this book.

Ease of Operations:

Effort required in integrating the system into operating environment may define case of operations. Ease of operations also talks about the help available to a user for doing any operation on the system (for example, online help, user manuals, operations manual). 'Ease of' operations' is different from "Ease of use' where the former considers user experience while using the system, while the latter consider show fast the system working knowledge can be acquired.

Reliability:

Reliability means that the system will perform its intended functions correctly over an extended time. Consistent results are produced again and again, and data losses are as less as possible in a reliable system. Reliability testing may be a base of "Build Verification Testing (VT)' where test manager tries to analyse whether system generates consistent results again and again.

Authorisation:

The data is processed in accordance with the intents of the user management. The authorisation may be required for highly secured processes which deal with huge sum of money or which have classified information. Applications dealing with classified information may need authorisation as the sensitivity of information is very important.

File Integrity:

Integrity means that data will remain unaltered in the system and whatever goes inside the system will be reproduced back in the same way. Accepting data in correct format, storing it in the same way, processing it in a way so that data does not get altered and reproducing it again and again are covered under file integrity. Data communication within and from one system to another may also be considered under the scope of file integrity.

Audit Trail Audit trail talks about the capability of software to substantiate the processing that has occurred. Retention of evidential information about the activities done by users for further reference may be maintained.

Continuity of Processing:

Availability of necessary procedures, methods and backup information to recover operations, system, data, etc. when integrity of the system is lost due to problems in the system or the environment define continuity of processing. Timeliness of recovery operations must be defined by the customer and implemented by developing organisations.

Service Levels:

Service levels mean that the desired results must be available within the time frame acceptable to the user, accuracy of the information must be

reliable, and processing completeness must be achieved. For some applications in Business, service level definition may be a legal requirement. Service level may have direct relationship with performance.

Access Control:

The application system resources will be protected against accidental or intentional modification, destruction, misuse or disclosure by authorised as well as unauthorised people. Access control generally talks about logical access control as well as physical access control for information, asses. etc.

2.12 SOFTWARE QUALITY MANAGEMENT

Quality management approaches talk about managing quality of a product or service using systematic ways and methods of development and maintenance. It is much above achieving quality factors as defined in software requirement specifications. Quality management involves management of all inputs and processing to the processes defined so that the output from the process is as per defined quality criteria. It talks about three levels of handling problems, namely,

Correction:

Correction talks about the condition where defects found in the product or service are immediately sorted and fixed. This is a natural phenomenon which occurs when a tester defines any problem found during testing. Many organisations stop at fixing the defect though it may be defined as corrective action by them. Responsibility of finding and fixing defects may be given to a line function. This is mainly a quality control approach.

Corrective Actions:

Every defect needs an analysis to find the root causes for introduction of a defect in the system. Situation where the root cause analysis of the defects is done and actions are initiated to remove the root causes so that the same defect does not recur in future is termed as corrective action, Corrective action identification and implementation is a responsibility of operations management group. Generally, project leads are given the responsibilities of initiating corrective actions. This is a quality assurance approach where process-related problems are found and resolved to avoid recurrence of similar problems again and again.

Preventive Actions:

On the basis of root causes of the problems, other potential weak areas are identified. Preventive action means that there are potential weak areas where defect has not been found till that point, but there exists a probability of finding the defect. In this situation, similar scenarios are checked and actions are initiated so that other potential defects can be eliminated before they occur. Generally, identification and initiation of preventive actions are a responsibility of senior management. Project

managers are responsible for initiating preventive actions for the projects. This is a quality management approach where an organisation takes preventive action so that there is no defect in the first place.

Quality management is a set of planned and systematic activities which ensures that the software processes and products conform to requirements, standards and processes defined by management, customer, and regulatory authorities. The output of the process must match the expectations of the users.

2.13 WHY SOFTWARE HAS DEFECTS?

One very important question about a product is, *Why there are defects in the product at all?'. There is no single answer to this question. After taking so much precaution of defining and implementing the processes, doing verification and validation of each artifact during SDLC, yet nobody can claim that the product is free of any defects. In case of software development and usage, there are many factors responsible for its success/failure. Few of them are,

- There are huge communication losses between different entities as requirements get converted into the actual product. Understanding of requirements is a major issue and majority of the defects can be attributed to this.
- Development people are more confident about their technical capabilities and do not consider that they can make mistakes. Sometimes self-review and/or peer review does not yield any defects.
- Requirement changes are very dynamic. As the traceability matrix is not available, impact analysis of changing requirements becomes heuristic.
- Technologies are responsible for introducing few defects. There are many defects introduced due to browsers, platforms, databases, etc. People do not read and understand release notes, and consequences of failure are attributed to technologies.
- Customer may not be aware of all requirements, and the ideas develop as the product is used. Proto typing is used for clarifying requirements to overcome this problem to some extent.

2.14 PROCESSES RELATED TO SOFTWARE QUALITY

Quality environment in an organisation is established by the management. Quality management is a temple built by pillars of quality. Culture of an organisation lays the foundation for quality temple. Every organisation has different number of tiers of quality management system definition, Figure 2.6 shows a relationship between vision, mission(s), policy(ies), goal(s), objective(s) strategy(ies) & values of organisation.

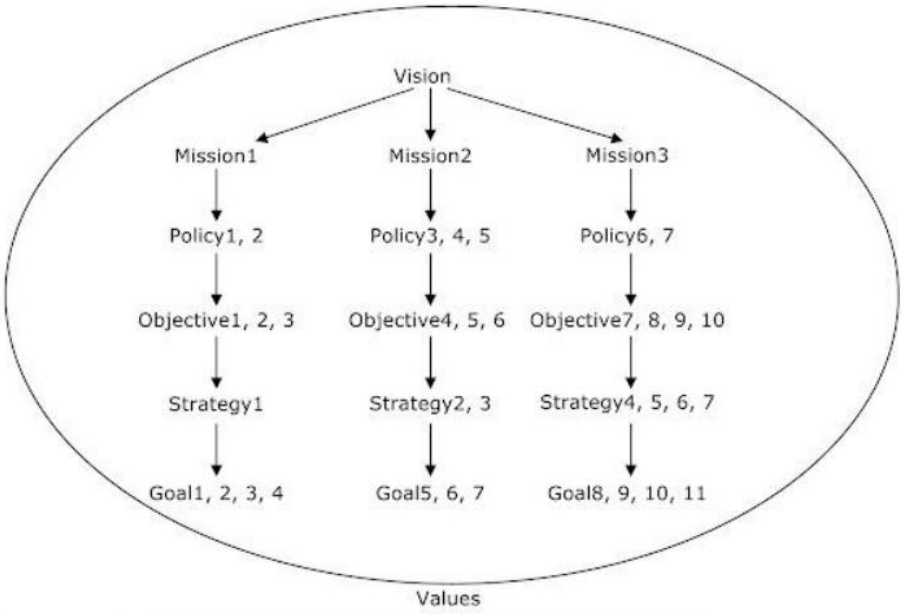


Fig. 2.6 Relationship between Vision, Mission(s), Policy(ies), Objective(s), Strategy(ies), Goal(s) and Values

2.14.1 Vision:

The vision of an organisation is established by the policy management. Vision defines in brief about what the organisation wishes to achieve in the given time horizon. 'To become a billion-dollar company within3 years' can be a vision for some organisations. Every organisation must have a vision statement, clearly defining the ultimate aim it wishes to achieve with respect to time span.

2.14.2 Mission:

In an organisation, there are several initiatives defined as missions which will eventually help the organisation realise its vision. Success of all these missions is essential for achieving the organisation's vision. The missions are expected to support each other to achieve the overall vision put by management. Missions may have different lifespans and completion dates.

2.14.3 Policy:

Policy statement talks about a way of doing business as defined by senior management. This statement helps employees, suppliers and customers to understand the thinking and intent of management. There may be several policies in an organisation which define a way of achieving missions. Examples of policies may be security policy, quality policy, and human resource development policy.

2.14.4 Objectives:

Objectives define quantitatively what is meant by a successful mission. It defines an expectation from each mission and can be used to measure the success/failure of it. The objectives must be expressed in numerals along

with the time period defined for achieving them. Every mission must have minimum one objective.

2.14.5 Strategy:

Strategy defines the way of achieving a particular mission. It talks about the actions required to realize the mission and way of doing things. Policy is converted into actions through strategy. Strategy must have a time frame and objectives along with goals associated with it. There may be an action owner to lead the strategy.

2.14.6 Goals:

Goals define the milestones to be achieved to make the mission successful. For a mission to be declared as successful/ failure at the end of the defined time frame in terms of whether the objectives are achieved or not, one needs a milestone review to understand whether the progress is in proper direction or not. Goals provide these milestone definitions.

2.14.7 Values:

Values can be defined as the principles, or way of doing a business as perceived by the management. "Treating customer with courtesy' can be a value for an organisation. The manner in which the organisation and management think and behave, is governed by the values it believes in.

2.15 QUALITY MANAGEMENT SYSTEM STRUCTURE

Every organisation has a different quality management structure depending upon its need and circumstances. Generic view of quality management is defined below. Figure 2.7 shows a structure of quality management system in general.

Fig. 2.7 Quality Management System of a typical organisation

2.15.1 1st Tier-Quality Policy:

Quality policy sets the wish, intent and direction by the management about how activities will be conducted by the organisation. Since management is the strongest driving force in an organisation, its intents are most important. It is a basic framework on which the quality temple rests.

2.15.2 2nd Tier-Quality Objectives:

Quality objectives are the measurements established by the management to define progress and achievements in a numerical way. An improvement in quality must be demonstrated by improvement in achievements of quality factors (test factors) in numerical terms as expected by the management. The achievements of these objectives must be compared with planned levels expected and results and deviations must be acted upon.

2.15.3 3rd Tier-Quality Manual:

Quality manual, also termed as policy manual is established and published by the management of the organisation. It sets a framework for other process definitions, and is a foundation of quality planning at organisational level.

2.16 PILLARS OF QUALITY MANAGEMENT SYSTEM

Top part of the quality temple is built upon the foundation of following pillars.

2.16.1 Quality Processes/Quality Procedures/Work Instructions:

Quality processes, quality procedures, work instructions, methods, etc, are defined at an organisation eye by the functional area experts, and at project and function level by the experts in those areas separately. Organisation level processes act as an umbrella, whereas project and function level processes are in the purview of these top-level process definitions. Organisation level set of processes may differ from the process definition for different projects and functions. It is also defined as quality planning at project level. Quality procedures must be in sync with the tone established by quality manual at an organisation level.

2.16.2 Guidelines and Standards:

Guidelines and standards are used by an organisation's project team for achieving quality goals for the products and services delivered to customers. Many a times, guidelines defined by customers are termed as standards for the project, as the project team takes the recommendations by customers as mandatory. Difference between a guideline and a standard is defined as, shown in Table 2.3.

Table 2.3 Difference Between Guidelines and Standards

Guidelines	Standards
• Guidelines are suggested ways of doing things. They are made by experts in individual fields. • Guidelines may be overruled and there is no issue if somebody does not follow it. • Guidelines may or may not be written. Generally, it is recommended that one must write the guidelines to capture the tacit knowledge	• Standards are mandatory ways of doing things. These are also described by experts in respective fields. • Overruling of standards is a punishable offence. It may lead to nonconformance during reviews and audits. • Standards must be written to avoid any misunderstanding or loss of communication
Guidelines and standards may need revisions from time to time. Revision must be done to maintain suitability over a time period.	

2.16.3 Formats and Templates:

Common formats and templates are used for tracking a project, function, and department information within an organisation. It creates same understanding across the board where outputs can be compared for the projects and functions. This also acts as a checklist to maintain consistency across the projects in the organisation. Formats and templates, if made compulsory, are considered as standards whereas if they are indicative or suggestive, they are considered as guidelines. Generally, templates are mandatory while formats are suggestive in nature.

2.17 IMPORTANT ASPECTS OF QUALITY MANAGEMENT

Quality improvement is not an accident but a planned activity. An organisation must plan for improvement under the leadership of management and with employee participation.

2.17.1 Quality Planning at Organisation Level:

An organisation creates quality plan at the organisation level for achieving quality objectives, goals, its vision and missions. Quality planning includes establishing missions, policies and strategies at organization level along with objectives and goals to achieve the vision, It must set a framework for definition and implementation of good processes, practices, recruiting people, infrastructures, hardware and software, There should be an appraisal of quality achieved as against expected results at planned intervals, and actions must be initiated in case of any deviation.

2.17.2 Quality Planning at Project Level:

Projects should plan for quality at project level. These are generally strategic-level quality plans with details of responsibilities and actions,

Project plan must define all aspects of quality plan at project level, and may have a relation with the organisation's quality planning. The quality objectives of the project may be inherited from organisation level objectives or may be defined separately for the project. Project level objectives must be in syne with organisation level objectives.

2.17.3 Resource Management:

An organisation should use good inputs as required by quality planning so that the output of the processes match with the organisation's business plans. It includes people, machines, materials, and methods as the basic resources. Good processes and good technology need good people to perform the work and achieve planned results.

2.17.4 Work Environment:

Working environment is an important input for a good product and to achieve the organisation vision and missions, A good environment can help an organisation to build on its strength while a bad environment is a roadblock to achieving objectives, and can create problems in its mission of customer satisfaction. Many organisations employ special techniques of 'Working Climate Analysis' to understand its environment, and take actions for correcting it.

Work environment has two components, viz. external environment and internal environment. External environment is mainly a physical environment while internal environment is built in the heart and brain of individuals. Good team spirit and loyalty can be major factors contributing to organisational success.

2.17.5 Customer-Related Processes:

Customer-related processes must be analysed for their capability in servicing customers and achieving customer satisfaction. Requirement analysis, designing, project processes, product delivery and other processes related to the customer must be analysed for their capabilities, and corrective/preventive actions must be initiated if those are found to be inadequate. Only capable processes can yield results in a consistent way.

2.17.6 Quality Management System Document and Data Control:

Many organisations define quality management system on the basis of some quality standards/models. There may be some specific customer requirements for different standards and models which may help an organization in defining its own quality management system and following customer directives, Statistical process control and data management are essential for continuous improvement of processes.

2.17.7 Verification and Validation:

Verification and validation are performed by an organisation at each level of development and for each activity. Verification includes management reviews and technical reviews (such as code review and project planre

view) whereas validation involves different kinds of testing (such as unit testing and system testing, ensure that the work product meets the predefined acceptance criteria.

Verification:

Verification defines the processes used to build the product correctly. Verification is successful, if the processes and procedures are followed correctly as defined by the process framework and also, they are capable of giving results. Verification cannot directly ensure that the right product has been built but checks if it has been built in the right way.

Validation:

Validation ensures that the right product has been built. It involves testing a software product against requirement specifications, design specifications, and customer needs. The method followed for development may or may not be correct or capable, but the final output should be as per customer requirements.

2.17.8 Software Project Management:

Project management is a specific skill required in leaders of projects (for example, project manager). Project management involves planning, organising, staffing, directing, coordinating and controlling the project to satisfy customers by delivering the right product, on time, in the budgeted cost. Nowadays project managers have to perform different tasks like mentoring, guiding, and supporting rather than supervising people.

2.17.9 Software Configuration Management:

The work products are built and tested again and again. The defects found during verification and validation are corrected, and the work product undergoes further updations, integration and testing. Software configuration management involves creating work products, maintaining them, reviewing them by related stakeholders and updating them as and when required. Base lined work products are released for further development process.

2.17.10 Software Metrics and Measurement:

New methods of project management approach stress a need for measuring the product attributes and process capabilities to achieve the quality of final deliverables to customer. Metrics and measurement programs are established by an organisation to capture metrics data/process measurement to ensure that processes are followed correctly and are capable of giving desired outputs. The lacunae found in the processes as well as products can be taken as an input to initiate corrective actions.

2.17.11 Software Quality Audits:

Audits defined in dictionaries as an examination of accounts. Quality audits of software products and processes must be performed to analyse the quality of the products as well as processes used to make them. These can help in analysis of the situation and taking corrective/preventive actions at proper levels. Software quality audits are performed by qualified auditors at predefined levels. Audits can be categorised, as per the following:

The Auditing Agency Involved:

- Internal audits are conducted by the people internal to the organisation. It is also called as first-party audits.
- Customer audits are conducted by the customer or customer representatives. It also called as second-party audits.
- Certification audits are conducted by third-party certification bodies.

The Phase When the Audit is Conducted:

- Kick-off audits are conducted at the start of the activity, say at a project start.
- Phase-end audits at the end of a phase.
- Pre-delivery audits are conducted before giving any deliverable to a customer.
- Postmortem audits are conducted at the end of the activity, say a project closure.

Subject of the Audit:

- Product audits are conducted to ensure that planned arrangements for quality are achieved or not.
- Process audits are conducted to ensure that processes defined are followed or not.

2.17.12 Subcontract Management:

Suppliers are the stakeholders for the organisation and projects. An organisation must build a strong bond of relationship with its suppliers. It must analyse the inputs from suppliers to make sure that the organization gets proper inputs so that outputs can be managed as planned. There must be a methodology supported by values which stresses on developing a long-term relationship with the suppliers and avoids least purchase cost bids to total cost-benefit analysis. Suppliers may be supported to do statistical process control to reduce cost and delay in supply or service problems.

2.17.13 Information Security Management:

Information is one of the biggest assets of the organisation. An organisation must protect the information assets available in various databases and all information that has been developed, captured, and used. It must be able to use that information for its continuous/continual improvement. Tacit knowledge is very important from security of information. Information security is associated with three buzz words viz. confidentiality, integrity and availability.

2.17.14 Management Review:

Management must periodically review the status of different projects and functions to understand progress which in turn will help in achieving the organisation's vision. Management must plan for corrective and preventive actions if required as indicated by metrics. It must decide upon future business plans for improvements. Management reviews must be systematic and planned. Inputs, processes and outputs of management reviews must be defined.

Aim at Customer Satisfaction:

Everything done by an organisation is for achieving customer satisfaction. Management must devise a process of collecting customer feedback and periodically measure and monitor customer satisfaction It must initiate actions where the customer feedback is negative.

Have Measurable Objectives:

Measurable objectives are essential to track the progress made by the organisation. Qualitative objective may or may not be sufficient to ensure its achievement as the organisation achieves maturity, and the organisation should try to put quantitative objectives, at least for critical processes. Organisations must have a definition of goals along with objectives for continuous measurements.

Understand Requirements Accurately:

Understanding and defining customer requirements is the most challenging work for business analyst, system analyst and management. It needs to ensure that a developer must understand the requirements correctly and interpret the requirements into correct product. Requirement losses in terms of misinterpretation must be reduced. Implied requirements must be converted into defined requirements.

Implement P-D-C-A Cycle in Each Phase:

Plan-Do-Check-Act cycle of continual (continuous) improvement must be followed to ensure improvement in product and process quality. An organisation must plan for future, do the things as planned, check the actual results with the planned ones and take actions on deviations.

Detect and Remove Defects as Early as Possible, Prevention is Better Than Cure:

Software development, longer the defect remains in a system, more costly and more difficult it is to remove it. It would be always advantageous to detect the defect through review and testing process as early as possible. All defects must lead to process improvements so that defect recurrence is avoided.

Systematic Change Control and Version Control:

Any change in work product must go through the stages of draft, review, approve and baseline. Version control and labelling is used effectively during development process to identify work products. Many tools are available for managing change, though it can also be done manually. Configuration management is very important to give the right product to the customer.

Follow Easy to Use Standards/Conventions for Naming, Commenting, Coding and Documentation:

An organisation must define standards and guidelines which are very useful for normal developers and testers. Common standards and guidelines show the best way to do things. It spreads common understanding across the tears and reduces the chances of misinterpretation. People do not have to invent the wheel again and again but can use the experience of others by referring to these guidelines and standards. They must be very simple to understand and use.

Start with Compiling and Analysing Simple Metrics:

An organization must define simple metrics at the start which are useful for planning improvements and tracking them. The main purpose of metrics is to define improvement actions needed and measuring how much has been achieved.

2.18 SUMMARY

This chapter is based upon the foundation of the earlier chapter where we have seen quality perspectives. In this chapter, we have studied different constraints faced while building and testing a software product. It tries to link the relationship between quality improvement and productivity improvement. We have seen the cultural difference between quality conscious organisations ‘Q’ and less quality conscious organisations ‘q’.

This chapter elucidates various development models such as waterfall, iterative, incremental, and prototyping. New development methodologies like agile have also been introduced. We have also seen the criticality definition of different systems from the perspective of different stakeholders and how a tester must understand these system criticalities before devising testing.

We have learnt different types of requirements and problems faced while defining these requirements. We have listed all the quality factors (test factors)applicable for a system and how it affects testing.

Finally, we had dealt with quality management system as a generic model and seen that prevention is required rather than finding and fixing the defects.

2.19 EXERCISES

1) What are the constraints of software requirement specifications?
2) Explain relationship between quality and productivity
3) Explain the concept of ‘q’ organisations and ‘Q’ organisations.
4) Explain different development models.
5) How products are classified depending upon their criticality?
6) What are different types of requirements?
7) What problems are posed by the requirement stage?
8) What are the characteristics of good requirements?
9) Explain difference between expressed and implied requirements.
10) Explain difference between present and future requirements.
11) Explain difference between generic and specific requirements.
12) List and explain quality objectives (test objectives) applicable to software development and usage.
13) Explain generic quality management system structure for an organisation.

2.20 REFERENCES

- Software Testing and Continuous Quality Improvement by William E. Lewis, CRC Press, 3rd Edition, 2016.
- Software Testing: Principles, Techniques and Tools by M. G. Limaye, Tata McGraw Hill, 2017.
- Foundations of Software Testing by Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black, Cengage Learning, 3rd Edition.
- Software Testing: A Craftsman‟s Approach, Paul C. Jorgenson, CRC Press, 4th Edition, 2017.

- https://www.geeksforgeeks.org/software-engineering-software-quality-assurance
- https://www.simplilearn.com/software-quality-assurance-article
- https://www.javatpoint.com/software-quality-assurance

*****

UNIT - II

3

FUNDAMENTALS OF SOFTWARE TESTING

Unit Structure

3.0 OBJECTIVES

After studying this chapter the learner would be able to:

- Understand the basic terminologies used in testing.
- Understand different approaches to testing.
- Understand different Principles of Software testing.
- Differentiate between "TQM" and "Big Bang" Approaches.
- Differentiate between different types of testing.
- Recognise features of good testing.

3.1 INTRODUCTION

Software testing is the process of examining the correctness of the software product by considering all its attributes like reliability, scalability, portability, re-usability and usability. It also evaluates the execution of software components to find the bugs or defects. Software

development activities during a life cycle have corresponding verification and validation activities at each stage of development.

Software verification involves comparing a work product with processes, standards, and guidelines in simple words, to check that "are we building the software correctly?"

Software validation activities are associated with checking the outcome of developed product and the processes used with respect to standards and expectations of a customer, in other words, to check "are we building the correct system?" It is considered as a subset of software quality assurance activities though there is a huge difference between quality assurance and quality control. Cost of software verification as well as validation comes under appraisal cost, when one is doing it for the first time. When repeat verification/validation (such as retesting or regression testing) is done, it is defined as cost of failure.

There are many stages of software testing as per software development life cycle. It begins with feasibility testing at the start of the project, followed by contract testing and requirements testing, then goes through design testing and coding testing till final acceptance testing, which is performed by customer/user.

The main aim of the most testing methods is to systematically and actively locate the defects/errors in the program and repair them. Debugging is usually the next stage of testing. Debugging begins with some indication of the existence of an error.

3.2 NECESSITY OF TESTING

Software testing involves verification as well as validation activities such as checking the compliance of the artifacts and activities with respect to defined processes and standards, and executing the software program to ensure that it performs correctly as desired by the customer and expressed in requirement specification agreed between development team and customer, Testing involves finding the difference between actual behaviours with respect to the expected behaviours of an application. Testing helps people (developers, testers, managers, the users) to understand what the software does and how well it does. To ensure the quality of the software before it goes live, Testing is a necessity.

3.3 WHAT IS TESTING?

Testing is a term used to evaluate or examine or to check the software product that it is meeting the expected behaviour or not. In software development, Testing is used at key checkpoints in overall process to determine whether objectives are met or not. For example, in software development, product objectives are sometimes tested by product user representatives. When the design is complete, coding follows and the finished code is then tested at the unit or module level by each programmer; at the component level by the group of programmers

involved; and at the system level when all components are combined together. At early or late stages, a product or service may also be tested for usability.

3.4 FUNDAMENTAL TEST PROCESS

Testing must be a planned process rather than a onetime activity. It requires discipline to act upon it. The quality and effectiveness of testing are primarily determined by the quality of test processes used. The activities of testing can be divided into the following basic steps:

(i) Planning & Control

(ii) Analysis & Design

(iii) Implementation & Execution

(iv) Evaluating exit Criteria and Reporting

(v) Test Closure Activities

Let us discuss each of these steps in detail:

(i) Planning & Control:

Test Planning is the fundamental test process that involves defining the objective and goal of the testing process. It is a continuous process and performed in all life cycles. It helps to determine the scope and risks and the objectives of the testing. It helps to decide the overall approach of testing. It also helps in assigning different resources to different activities. It involves scheduling test analysis and design tasks, test implementation, execution and evaluation.

Control is the activity of comparing actual progress against the plan and reporting the status, including the deviations from the plan. It involves taking actions necessary to meet the mission and objectives of the project.

(ii) Analysis & Design:

It is the fundamental test process in which test cases and test conditions are defined. In this process, major tasks that are performed are reviewing the test basis i.e. information on which the test cases are based, such as requirements, design specifications, product risk analysis, architecture etc., identifying the test conditions based on the analysis of test items, writing test cases and identifying the necessary test data to support the test conditions and test cases, designing the test environment.

(iii) Implementation & Execution:

Test implementation and execution is a fundamental test process in which actual work is done. It involves actual running the specified test on a computer system either manually or by using an automated test tool. Test implementation involves major tasks like developing and prioritizing the test cases, creating test suites from test cases that help in efficient test

execution, re-executing the test cases that failed earlier to confirm the fix and also it also involves keeping the log of outcome of test execution. A test log is the status of the test case i.e. Pass or fail.

(iv) Evaluating exit Criteria and Reporting:

Evaluating exit criteria is a process that defines when to stop testing. It depends on various parameters like risk, functionality or coverage of code. It can also depend on the business risk, cost and time. It actually varies from project to project. Exit criteria starts when the maximum test cases are executed with certain pass percentage, bug rate falls below certain level and when the deadlines are met. Evaluating exit criteria must assess if more test are needed or if the exit criteria specified should be changed. It also involves writing a test summary report for stakeholders.

(v) Test Closure Activities:

It is the last process in the fundamental test process. In this the data is collected from completed test processes and test wares. It ensures that final deliverable has been delivered or not and also ensures that all incident reports have been resolved. It involves the finalisation and archiving the test wares such as scripts, test environments etc. for later reuse. Test closure activity also includes handing over the test ware to the maintenance organisation so that they give support to the software. It also includes the evaluation of how the testing went and learn lessons for the future releases and projects.

3.5 PSYCHOLOGY OF TESTING

In software testing, psychology plays an extremely important role. It is one of the factors that stays behind the scene but has a great impact on the end result. It can be categorised majorly into three sections:

(i) Mind-set of Developers and Testers:

SDLC is a combination of various activities which are performed by different individuals using their expertise and knowledge. It is an unknown fact that for successful development of software individuals having different skill and mind set are required. For e.g. as we all know testing and analysing the software is much more different than the developing or programming so is the need for the right individuals with right expertise and right mind set to develop the software with unique features and superior quality. Moreover, the change in the development and the testing techniques and methodologies has made this as a necessary requirement.

(ii) Communication in a constructive manner:

In any process whether technical or non-technical requires constant communication between team members to successfully achieve the goals. Communicating in a polite and courteous manner can help build strong relationship and avoid misunderstandings. This goes well with the tester,

finding errors and reporting them can be an achievement for him but is merely an inconvenience for programmers, designers, developers and other stake holders of the project. So, it is important for the testers to impart defects and failures as objectively and politely so as to avoid upsetting and hurting other members related to the project.

(iii) Test Independence:

The importance of independent software testing is comparatively higher than self-testing or group testing as it is more effective in finding and detecting bugs and defects. This type of testing is done by individuals who are not related to the project directly or are from different organisation and are hired mainly for testing the quality as well as the effectiveness of the developed product.

3.6 HISTORICAL PERSPECTIVE OF TESTING

The concept of independent testing did not prevail during the initial days of software development. It was believed that whatever the developers were doing was the best way of producing the product and the customer was expected to use it as it is. If there were any problems reported by customer/users, they would be fixed by the developers, and the application would be given again to customer/users. The primary responsibility of testing was with customers/users (and not with developers) during the production phase.

Glenford Myer introduced software testing as a separate phase in software development life cycle. According to him, software testers were expected to test software with all the possible combinations. The main intention was to create a software which would never fail in production. Testers were expected to have an attitude to break the software so that it would be eventually corrected and would never fail during use. This approach separated debugging from testing, and an independent testing community was created. Different phases of software testing evolution as a separate discipline in software development activities were followed in a cycle.

3.6.1 Debugging-Oriented Testing:

During the initial phase, software testing was considered as a part of software development. Developers were expected to perform debugging on the application, which they were building. Tests were not documented, and were mainly done in a heuristic way. Generally, testing was completely ‘positive testing’ to see whether the implementation was working correctly or not.

3.6.2 Demonstration-Oriented Testing:

In this phase, there was an introduction of software testers independent of development activity. Their main aim was to show to customer/users that the software really works. Although, this phase was much advanced than the initial phase of debugging, yet the approach was still positive testing only. The approach was oriented towards demonstration that the software

could do something which was expected by the customer/users. Test cases were generally derived from the requirement statements which were oriented towards successful demonstration.

3.6.3 Destruction-Oriented Testing

This approach was the basis of Glenford Myer's theory of software resting. As per this approach, it was not sufficient to only test the software positively but users must also be protected from any conceivable failure of application. The tester's responsibility changed from 'proving that software works under normal conditions' to 'proving that software does not fail at some abnormal instances'. Often, the testers were too imaginative in breaking the software, and defects for which there were no feasibility of happening were reported as defects. This phase was quiet frustrating to software developers -testers were considered as demons and testing was considered as a hurdle to be passed before delivering the application to the customer.

3.6.4 Evaluation-Oriented Testing:

Evaluation-oriented testing is executed nowadays at many places of software development where the product as well as process of software development is evaluated. It corresponds to the testing process definition where software is evaluated against some fixed parameters derived from quality factors (test factors). Quality factors/test factors are introduced in this phase as a part of customer requirements. It is believed that there is no possibility of software without any defect, but some level of defect may be acceptable to the customer. This approach also refers to level of confidence given to customer that application will work as expected by the user. The confidence level is determined and application is evaluated against it. Confidence-level expectations are linked with cost of testing.

3.6.5 Prevention-Oriented Testing:

Prevention-based testing is done in some highly matured organisations while for many others, the concept still utopia. Testing is considered as a prevention activity where process problems are used to improve it so that defect-free products can be produced. Every defect found in testing is considered as process lacunae, and efforts are initiated to improve the processes of development. This reduces dependency on testing as a way to improve the quality of software. This helps in reducing the cost by producing right product at the first time.

3.7 DEFINITION OF TESTING

Testing is defined as 'execution of a work product with intent to find a defect'. The primary role of software testing is not to demonstrate the correctness of software product, but to expose hidden defects so that they can be fixed. Testing is done to protect the common users from any failure of system during usage.

This approach is based on the assumption that any amount of testing cannot show that software product is defect free. If there is no defect found during testing, it can only show that the scenario and test cases used for testing did not discover any defect. From user's point of view, it is not sufficient to demonstrate that software is doing what it is supposed to do. This is already done by system architects in system architecture design and testing, and by developers in code reviews and unit testing. Testers are involved mainly to ensure that the system is not doing what it is not supposed to do. Their work includes assurance that the system will not be exposed to any major risks of failure when a normal user is using it. Some people call this approach as negative approach of testing. This negative approach is built upon few assumptions and risks for the software being developed and tested. These assumptions and risks must be documented in the test plan while deciding test strategy or test approach.

3.7.1 Why Testing Is Necessary?:

One may challenge testing activities by asking this question- 'If any level of testing cannot declare that there is no defect in the product, then why it is required at all?' In normal life, we find highly qualified and experienced people involved in each stage of development from requirement gathering till acceptance of software. Finding a defect in software is sometimes considered as challenging the capabilities of these people involved in development phases. Testing is necessary due to the following reasons.

- Understanding of customer requirements may differ from person to person. One must challenge the understanding at each stage of development, and there must be some analysis of customer expectations. Approach-related problems may not be found when there is no detail analysis by another person not involved emotionally with development. Everything is considered as 'OK' unless there is an independent view of a system.

- Development people assume that whatever they have developed is as per customer requirements and will always work. But it is imperative to create real-life scenario and undertake actual execution of a product at each level of software building (including system level) to assess whether it really works or not.

- Different entities are involved in different phases of software development.

Their work may not be matching exactly with each other or with the requirement statements. Gaps between requirements, design and coding may not be traceable unless testing is performed in relation to requirements.

- Developers may have excellent skills of coding but integration issues can be present when different units do not work together even though they work independently. One must bring individual units together and make the final product, as some defects may be possible when the sources are developed by people sitting at different places.

- There is a possibility of blindfold and somebody has to work as the devil's representative. Every person feels that what he/she has done is perfect and there is no chance of improvement. Testers have to challenge each assumption and decision taken during development.

3.8 APPROACHES TO TESTING

There are many approaches to software testing defined by the experts in software quality and testing, the approaches may differ significantly as per customer requirements, type of the system being developed as well as management thinking about software development life cycle followed by software, type of project, type of customer, and maturity of development team. These approaches form the part of testing strategy. Few of them are discussed below.

3.8.1 Big Bang Approach of Testing:

Characteristics of 'Big bang' approach involve testing software system after development work is completed. This is also termed 'system testing' or final testing done before releasing software to the customer for acceptance testing. This testing is the last part of software development as per waterfall methodology. Big bang approach has main thrust on black box testing of software to ensure that the requirements as defined and documented in requirement specifications and design specifications are met successfully. Testing done at the end of development cycle may show the defects pertaining to any phase of development such as requirements, design, and coding. Roughly saying, the phase-wise defect origination follows the trend shown in Table 3.1.

Table 3.1 Phase-wise defect distribution

Development Phases	Percentage of defects
Requirements	58
Design	35
Coding	5
Other	2

In case of big bang approach, software is tested before delivery using the executable or final product. It may not be able to detect all defects as all permutations and combinations cannot be tested in system testing due to various constraints like time. In such type of testing, one may find a cascading effect or camouflage effect, and all defects may not be detected. It may discover the failures but cannot find the problems effectively. Sometimes, defects found may not be fixed correctly as analysis and defect fixing can be a problem.

3.8.2 Total Quality Management Approach:

If there is a process definition for testing software, and these processes are optimised and capable, no (less) defects are produced and no (few) undetected defects are left in the software when it is delivered to the

customer. Defect removal costs are approximately 10 times more after coding than before coding. This is a cost associated with fixing the problems belonging to requirements, design, coding, etc. If the defect is not detected earlier but found in acceptance testing or further down the line during warranty, it may be much more costly. Defect removal cost would be 100 times more during production (at user site) than before coding. This may involve deploying people at customer site, loss of goodwill, etc. which is a part of failure cost.

3.8.3 Total Quality Management (Tqm) As Against Big Bang Approach:

Figure3.1 is a very famous cost triangle defined by TQM. If the organisation has very good processes defined which are optimised and capable, and can produce consistent results again and again, then it gives advantage in productivity and effectiveness in development. It can reduce cost of production significantly.

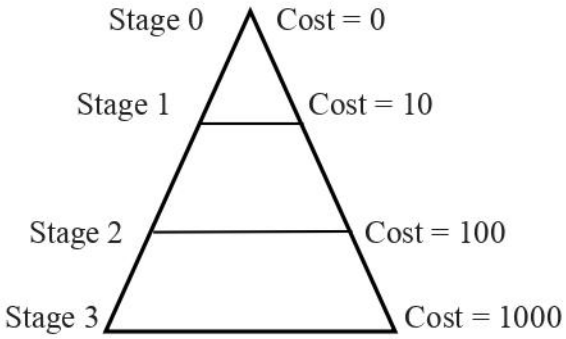


Fig.3.1 TQM Cost Triangle

Stage 0:

There may be a stage of maturity in an organisation where no verification/validation is required to certify the product quality. The cost involved is

'Zero' or the benefits derived by investment in process definition, optimisation and deployment make quality free. There is a famous saying 'quality is free'.

Theoretically, if the processes are optimised, there is no need of verification/validation as the defects are not produced at all.

Stage 1:

Even if some defects are produced during any stage of development in such quality environment, then an organisation may have very good verification processes which may detect the defects at the earliest possible stage and prevent defect percolation. There will be a small cost of verification and fixing, but stage contamination can be saved which helps in finding and fixing the defects very fast. This is an appraisal cost

represented by '10'. This is the cast which reduces the profitability of the organisation due to scrap, rework and reverification.

Stage 2:

If some defects escape the verification process, still there are capable validation processes for filtering the defects before the product goes to a customer.

Cost of validation and subsequent defect fixing is much higher than verification. This cost is represented by '100'. One may have to go to the stage where defect was introduced in the product and correct all the stages from that point onward till the defect-detection point. The cost is much higher, but till that point of time the defect has not reached the customer, it may not affect customers' feelings or goodwill.

Stage 3 At the bottom of the pyramid, there is the highest cost associated with the defects found by customer during production or acceptance testing. This is represented by '1000' showing that cost paid for paid for fixing such defect is huge. There may be customer complaints, selling under concession, sending people onsite for fixing defects in front of the customer, loss of goodwill, etc. This may result into premature closure of relationship and bad advertisement by the customer.

3.8.4 Tqm in Cost Perspective:

Total quality management (TQM) aims at reducing the cost of development and cost of quality through continual improvement. Often, it is termed 'Quality is free'. It means that the cost of quality must repay much more than what has been invested. TQM defines the cost incurred in development and quality into three parts as follows.

Green Money/Cost of Prevention:

Green money is considered as an investment by the organization in doing quality work. It is a cost spent in definition of processes, training people, developing foundation for quality, etc. It gives return on investment, and includes all prevention-based costs. If an organisation has defined and optimised processes, trained people, and fixed guidelines and standards for doing work which are allowed, then the return on such investment can be seen in terms of less inspection and testing, and higher satisfaction with repeat orders. This improves profitability of an organisation.

Blue Money/Cost of Appraisal:

Blue money is a cost incurred by the organisation during development, in the form of first-time review testing which gets returned in future. It does not earn profit, but it is an essential part of development process to ensure the process capability. All appraisal techniques used during SDLC such as first-time verification or validation are considered as blue money. First-time testing helps in certifying that nothing has gone wrong and work product can go to the next stage. In initial phases, the cost of appraisal increases, but as the organisational process maturity increases, this cost

must go down as fewer samples are required to prove the correctness of the process of making software.

Red Money/Cost of Failure:

Red money is a pure loss for the organisation. It involves money lost in scrap, rework, sorting, etc. It also represents loss due to various wastes produced during development life cycle, and directly reduces the profit for the organisation and the customer may not pay for it. As the investment or green money increases, failure cost must go down. All cost incurred in reinspection, retesting, and regression testing represent cost of failure.

3.8.5 Characteristics of Big Bang Approach:

Big bang approach talks about testing as the last phase of development. All the defects are found in the last past of rework can be huge.

- Testing is the last phase of development life cycle when everything id finalised. Heavy costs and efforts of testing are seen at the end of software development life cycle representing 'Big bang' approach. Most of the testing is concentrated in this phase only, as there are no previous verification or validation activities spread during the development phases.

- Big bang approach is characterized by huge rework, retesting, scrap, sorting of software components, and programs after the complete software is built. There may be special teams created to fix defects found in final testing. Big bang works only on correction, and there are no corrective and preventive actions and process improvements arising from their defects. The processes remain immature and every time, defect fixing becomes an invention of the wheel.

- Regression testing reveals many issues, as correction may not be correct and may introduce some defects in the product. There are many interdependencies between various entities while building software product, and these may get affected adversely. When some areas in the software units are touched for correction or fixing of defect, dependent components may get affected in a negative way. These may be termed 'regression defects'.

- All requirements and designs cannot be covered in testing. As the schedule of delivery in fixed and development generally lags behind the schedule, thus huge adhoc, exploratory, monkey, and random testing are done in this part of testing. Testing is done in a hurry, and many iterations of defect fixing and testing are done in the shortest possible time. Some defects may flow to customer an 'known defects' or sometimes they are not declared at all.

- The major part of software build never gets tested as coverage cannot be guaranteed in random testing. Software is generally tested by adhoc methods and intuition of testers. Generally, positive testing is

done to prove that software is correct which represents second level of maturity.

Organisations following big bang approach are less matured and pay a huge cost of failure because of several retesting and regression testing along with defect-fixing cycles. Success is completely dependent on good development, testing team and type of' customer. The following can be observed.

- Verification activities during software development life cycle can find out about two-third of the total number of defects found. The cost involved in fixing such defects is much less than any other way of finding the defects and fixing them. There is less stage contamination as defects are prevented from progressing from one stage of development to another.

- Validation in terms of unit testing can find out about three-fourth of the remaining defects. Defects are found in the units and fixed at that point itself so that they do not occur further down the line. It reduces the chances of defects being found in system testing. Unit testing validates an individual unit.

- Validation in terms of system testing can find out about 10% of the total number of defects. System testing must be intended to validate system-level requirements along with some aspect of design. Some people term system testing as certification testing while some people term acceptance testing as certification testing. If the exit criteria of system testing (acceptance criteria by customer) are met, system may be released to the customer.

Remaining defers (about 5-10%) go to the customer, unless the organisation makes some deliberate efforts to prevent them from leaking to production phase. Big bang approach may not be useful in preventing defects from going to the customer, as it can find only 5% of the total defects present in the product. In other terms, theoretically, to achieve the effectiveness of life-cycle testing, one may need about 18cycles of system testing. This can prove to be a costly affair.

3.9 POPULAR DEFINITIONS OF TESTING

Let us try to define ‘software testing’ keeping the background of Big bang approach in mind. All definitions of testing indicate that testing is a life-cycle operation and not the activity at the end of development phase. No definition of testing can really cover all aspects of testing. Hence, no definition is complete but indicates a part of what software testing is.

3.9.1 Traditional Definition of Testing:

There can be many definitions of testing pertaining to different instances. Few of them are as follows.

- Testing is done to establish confidence that the program does what it is supposed to do. It generally covers the functionalities and features expected in the software under testing. It covers only positive testing.

- Testing is considered as any activity aimed at evaluating an attribute or capability of a program or system with respect to user requirements. To some extent, this definition may be considered as correct, as number of defects found in testing is directly proportional to number of defects remaining in the system.

- Testing is a process of demonstrating that errors are not present in the product. This approach is used in acceptance testing where if the application meets acceptance criteria, then it must be accepted by the customer.

- Testing gives number of defects present which indirectly gives a measurement of software quality. More number of defects indicate bad software and bad processes of development.

- Testing is done to evaluate the program or system used for making software. As we consider that defects are introduced due to incapable processes, testing may be used to measure process capability to some extent.

- Testing is used to confirm that a program performs its intended functions correctly. Intended functionality may be defined from requirement specifications.

If testing is defined as a process, then it is designed to,

- Prove that the program is error free or there is no defect present

- Establish that the software performs its functions correctly and is fit for use

- Establish that all expectations of functionalities are available

Testing may not be any of these certification activities. If the goal of testing is to prove that an application works correctly, then the tester should subconsciously work towards this goal, choosing test data that would prove that the system is working correctly. The reverse would be true if the goal of testing is to locate defects so that eventually these would be corrected. Test data should be selected with an eye towards providing the test cases that are likely to cause product failure.

3.9.2 What Is Testing?:

Let us try to define what is meant by testing with this background. Testing is completely guided by software requirements specifications and design specifications, and supported by test strategy and test approach depending on assumptions and risks of development, testing and usage. Testing process may include the following.

- An activity of identification of the differences between expected results and actual results produced, during execution of software application. Difference between these two results, suggests that there is a possibility of defect in the process and/or work product. One must note that this may or may not be a defect.
- Process of executing a program with the intention of finding defects. It is expected that these defects may be fixed by the development team during correction, and the root causes of the defects are also found and closed during corrective actions. This can improve development process.
- Detecting specification-related errors and deviations of working application with respect to the specifications. Requirement mismatches and misinterpretation must be detected by testing.
- Establish confidence that a program does what it is supposed to do. This defines the confidence level imparted by software testing to a customer that the software will work under normal conditions. The expectation of confidence level is a function of depth and width of software testing.
- Any activity aimed at evaluating an attribute of a program or software. Acceptance testing is an activity defining whether the software has been accepted or not.
- Measurement of software quality in terms of coverage of testing (in terms of requirements, functionality, features, and number of defects found) can give information about confidence level imparted to a customer.
- Process of evaluating processes used in software development. Every failure/defect indicates a process failure. This can be used to improve development processes.
- Verifying that the system satisfies its specified requirements as defined, and is fit for normal use. Requirements may be elicited with the help of the customer.
- Confirming that program performs its intended functions correctly.
- Testing is the process of operating a system or component under specified conditions, observing and recording the results of such processing, and evaluating some aspect of system or component on the basis of testing.
- Software testing is the process of analysing a software item to detect the difference between existing and required conditions, and to evaluate the feature of the software item.

We have previously discussed about different stakeholders and their interests in software development and testing. Let us try to analyse the expectations or views of different stakeholders about testing.

3.9.3 Manager's View of Software Testing:

The senior management from development organisation and customer organisation have the following views about testing the software product being developed.

- The product must be safe and reliable during use, and must work under normal as well as ad verse conditions when it is actually used by the intended users.
- The product must exactly meet the user's requirements. These may include implied as well as defined requirements.
- The processes used for development and testing must be capable of finding defects, and must impart the required confidence to the customer.

3.9.4 Tester's View of Software Testing:

Testers have different definitions about software testing, as mentioned below.

- The purpose of testing is to discover defects in the product and the process related to development and testing. This may be used to improve the product and processes used to make it.
- Testing is a process of trying to discover every conceivable fault or weakness in a work product so that they will be corrected eventually. Random testing sometimes becomes too imaginative, and unconceivable defects may be found.

3.9.5 Customer’s View of Software Testing:

Customer is the person or entity who will be receiving using the product and will be paying for it. Testers are considered as the representatives of the customer in system development.

- Testing must be able to find all possible defects in the software, along with related documentation so that these defects can be removed. Customer must be given a product which does not have defects (or has minimum defects).
- Testing must give a confidence that software users are protected from any unreasonable failure of a product. Mean time between failures must be very large so that failures will not occur, or will occur very rarely.
- Testing must ensure that any legal or regulatory requirements are compiled during development.

Testing is an activity which is expected to reduce the risk of software's failure in production. All stakeholders have many expectations from testing. Let us try to analyse the meaning of a successful tester.

3.9.6 Objectives of Testing:

To satisfy the definition of testing given earlier, testing must accomplish the following things.

- Find a scenario where the product does not do what it is supposed to do. This is deviation from requirement specifications.
- Find a scenario where the product does things it is not supposed to do. This includes risk.

The first part refers to specifications which were not satisfied by the product while the second part refers to unwanted side effects while using the product.

3.9.7 Basic Principles of Testing:

The basic principles on which testing are based are given below.

- Define the expected output or result for each test case executed, to understand if expected and actual output matches or not. Mismatches may indicate possible defects. Defects may be in product or test cases or test plan.
- Developers must not test their own programs. No defects would be found in such kind of testing as approach-related defects will be difficult to find. Development teams must not test their own products. Blindfolds cannot be removed in self-testing.
- Inspect the results of each test completely and carefully. It would help in root cause analysis and can be used to find weak processes. This will help in building processes rightly and improving their capability.
- Include test cases for invalid or unexpected conditions which are feasible during production. Testers need to protect the users from any unreasonable failure so that one can ensure that the system works properly.
- Test the program to see if it does what it is not supposed to do as well as what it is supposed to do.
- Avoid disposable test cases unless the program itself is disposable. Reusability of test case is important for regression. Test cases must be used repetitively so that they remain applicable. Test data may be changed in different iterations.
- Do not plan tests assuming that no errors will be found. There must be targeted number of defects for testing. Testing process must be capable of finding the targeted number of defects.

- The probability of locating more errors in any one module is directly proportional to the number of errors already found in that module.

3.9.8 Successful Testers:

The definition by testers about testing talks about finding defects as the main intention of testing. Testers who find more and mare number of defects are considered as successful. This gives some individuality to testing process which talks about ability of a tester to find a defect. There is a difference between executing a test case and finding the defect. This needs an ability to look for detailing, problem areas, and selection of test data accordingly.

- Testers must give confidence about the coverage of requirements and functionalities as defined in test plan.
- Testers must ensure that user risks are identified before deploying the software in production.
- Testers must conduct SWOT analysis of the software and processes used to make it. This can help in strengthening the weaker areas and the processes responsible for defects so that the same problems do not recur.

3.9.9 Successful Test Case:

Testing is a big investment and justifies its existence, if it catches a defect before going to the customer. Every defect caught before delivery means the probability of finding a defect by a customer is reduced. If testing does not catch any defect, it is a failure of testing and a waste for the organisation as well as customer. Testing involved in software development life cycle starts from requirements, goes through design, coding, and testing till the application is formally accepted by user/customer.

3.10 TESTING DURING DEVELOPMENT LIFE CYCLE

Let us discuss life cycle phase and testing associated with it. This discussion is based on the consideration that development methodology follows waterfall cycle/model.

Requirement Testing:

Requirement testing involves mock running of future application using the requirement statements to ensure that requirements meet their acceptance criteria. This type of testing is used to evaluate whether all requirements are covered in requirement statement or not.

This type of testing is similar to building use cases from the requirement statement. If the use case can be built without making any assumption about the requirements, by referring to the requirements defined and documented in requirement specification documents, they are considered to be good. The gaps in the requirements may generate queries or

assumptions which may possibly lead to risks that the application way not performs correctly. Gaps also indicate something as an implied requirement where the customer may be contacted to get the insight into business processes. It is a responsibility of business analyst to convert (as many as possible), implied requirements to expressed requirements. Target is 100%, though it is difficult to achieve.

Requirement testing differs from verification of requirements. Verification talks about review of the statement containing requirements for using some standards and guidelines, while resting talks about dummy execution of requirements to find the consistency between them, i.e., achievement of expected results must be possible by requirements without any assumption. Verification of requirements may talk about the compliance of an output with defined standards or guidelines.

The characteristics of requirements verification or review may include the following.

- Completeness of requirement statement as per organisation standards and formats. It must cover all standards like performance and user interface expected by customer organisation.
- Clarity about what is expected by the users at each step of working while using an application. It must include the expected output by the customer. It may be in the form of error messaging, screen outputs, priority outputs, etc.
- Measurability of expected results, possibly in numerals, so that these results can be tested. Test case will have expected results which must satisfy measurement criteria defined. 'User friendliness' or 'fairly fast' can create confusion about requirements.
- Testability of the scenario defined in requirement statement is must. Some requirements like application must work 24 x 7 for 10 years may not be directly testable.
- Traceability of requirements further down the development life cycle must be ensured. Requirement traceability starts at requirement phase and gets populated as one goes down the life cycle.

Theoretically, each statement in requirement document must give atleast one functional/non-functional test scenario which may result into test cases. Requirements must be prioritised as 'must', 'should be' and 'could be' requirements. The customer is an entity to confirm requirement priority.

Requirement validation must define end-to-end scenario completely so that there is no gap. It must talk transactions involved and information transfer from one system to another.

Design Testing:

Design testing involves testing of high-level design (system architecture) as well as low-level design (detail design). High-level design testing covers mock running of future application with other prerequisites, as if it is being executed by the targeted user in production environment. This testing is similar to developing flow diagrams from the designs, where flow of information is tracked from start to finish. When the flow is complete, the design may be considered as good. Wherever the flow is not defined, or not clear about where it will lead to, there are defects with design which must be corrected. For low-level design, system requirements and technical requirements are mapped with the entities created in design to ensure adequacy of detail design.

Design verification talks about reviewing the design, generally by the experts who may be termed as subject-matter experts. It involves usage of standards, templates, and guidelines defined for creating these designs. Design verification ensures that designs meet their exit criteria.

- Completeness of design, in terms of covering all possible outcomes of processing and handling of various controls as defined by requirements.
- Clarity of Row of data within an application and between different applications which are supposed to work together in production environment.
- Tenability of a design which talks about software structure and structural testing.
- Traceability with requirements.
- Design toast covers all requirements.

Code Testing:

Code files, Tables, Stored procedures etc are written by developers as per guidelines, standards, and detail design specifications. In reality, developers do not implement requirements directly but they implement detail design as delivered by the designer. Code testing (unit testing) is done by using stubs/drivers as required. Code review is done to ensure that code files written are,

- Readable and maintainable in future. There are adequate comments available.
- Testable in unit testing.
- Traceable with requirements and designs. Anything extra as well as anything missing can be considered as a defect.
- Testable in integration and system testing.

- Optimised to ensure better working of software. Reusability creates a lighter system.

Test Scenario and Test Case Testing:

Test scenarios are written by testers to address testing needs of a software application. Test cases are derived from test scenarios which are related to requirements and designs. Test scenarios can be functional as well as structural, depending upon the type of requirement and design they are addressing.

- Test scenario should be clear and complete, representing end-to-end relationship of what is going to happen and also, the possible outcomes of such processing.
- Test scenarios should cover all requirements. Test scenarios may be prioritised as per requirement priorities.
- Scenarios should be feasible so that they can be constructed during testing.
- Test cases should cover all scenarios completely.
- Test scenarios and test cases must be prioritised so that in case of less time availability, the major part of the system (where priority is higher) is tested.

3.11 REQUIREMENT TRACEABILITY MATRIX

Some quality management models and standards prescribe complete traceability of a software application from requirements through designs and code files up to test scenario, test data, test cases and test results. Requirement traceability matrix is one way of doing the complete mapping for the software. One can expect a blueprint of an entire application using requirement traceability matrix.

Typical requirement traceability matrix is as shown in Table 3.2.

Table 3.2 Requirement traceability matrix

Require-ments	High-level Design	Low-level design	Code files/ Stored Proce-dures/TBLs	Test scenario	Test cases	Test results

3.11.1 Advantages of Requirement Traceability Matrix:

As discussed earlier, Requirement traceability matrix is a blueprint of software under development. All the agencies concerned with software can use it to understand the software in a better way. It may answer questions about what is being developed and how it will be implemented.

It helps in tracing if any software requirement is not implemented, or if there is a gap between requirements and design further down the line. It also helps to understand if any redundancy has been created in the application.

- Entire software development can be tracked completely through requirement traceability matrix.
- Any test-case failure can be tracked through requirements, designs, coding, etc.
- Any changes in requirements can be affected through entire work product upto test cases and vis-à-vis any test case failure can be traced back to requirements.
- The application becomes maintainable as one has complete relationship from requirement fill test results available.

3.11.2 Problems with Requirement Traceability Matrix:

Theoretically, all software's must have requirement traceability matrix, but in reality, most of the software's do not have it. The reasons are numerous; some of the prominent ones are listed below.

- Number of requirements is huge. It is very difficult to create requirement traceability matrix manually. For using some tools, one needs to invest money. Also, people may need to be trained for using tools.
- There may be one-to-many, many-to-one and many-to-many relationships between various elements traceability matrix, when we are trying to connect columns and rows of traceability matrix and maintaining these relationships need huge efforts.
- Requirements change frequently, and one needs to update the requirement traceability matrix whenever there is a change. Similarly designs, code and test cases may also change which will affect traceability matrix.
- Developing teams may not understand the importance of requirement traceability matrix, if development follows waterfall model, and during maintenance, it may be too late to create it. Incremental and iterative developments are the major challenges for maintaining traceability.
- A customer may not find value in it and may not pay for it.

3.11.3 Horizontal Traceability:

When an application can be traced from requirement through design and coding till test scenario and test cases upto test results, it is termed as horizontal traceability. On failure of any test case, we must be able to find which requirements have not been met. Any design which does not have

requirement. Introduces an extra feature which may be considered as defect. Similarly, when any requirement is not traceable to design, that requirement is not implemented at all. Same thing can happen in the relationship between design and coding, coding and test scenario, and also, test scenario and test case. When any entity can't be traced in forward direction, horizontal traceability is lost.

3.11.4 Bidirectional Traceability:

One must be able to go from requirements, designs, coding, and testing to reach the test result. Reverse must also be possible, where one may start from the result and go to requirements. One must be able to go in any direction from any point in traceability matrix. This is referred to as 'bidirectional traceability'; CMMi model mandates bidirectional traceability for all products.

3.11.5 Vertical Traceability:

Traceability explained above is called 'horizontal traceability' as it goes in horizontal direction, either forward or backward. Traceability may exist in individual column as the requirements may have some interdependencies between them, or these may be child and parent relationships. For achieving a requirement, the other child requirement must be achieved. One requirement may have several child requirements, while some child requirements may have several parent requirements. If these requirements are traced completely, it ensures vertical traceability. Similarly, design, coding, and testing may have a vertical traceability where there may be parent-child relationship and interdependence on different parts. Designs may have parent-child relationships, and coding may have 'called functions' and 'calling functions' traceability relationships.

3.11.6 Risk Traceability:

Some application development organisations also add references about the risks of failure faced by the application in Failure Mode Effect Analysis (FMEA). The risks are traced to requirements and mainly with design which defines control mechanism to reduce probability or impact or improves detection ability of a risk this helps the customer to identify which accident-prone zones are in the application and where the user is completely/ partially protected from failures. It also helps in identifying various types of controls that are designed and used. Typical risk traceability matrix is as shown in Table 3.3.

Table 3.3 Risk traceability matrix

Risk	High-level Design/Control	Low-level design/Control	Code files/ Stored procedures/TBLs	Test scenario	Test cases	Test results

3.12 ESSENTIALS OF SOFTWARE TESTING

Software testing is a disciplined approach. It executes software work products and finds defects in it. The intention of software testing is to find all possible failures, so that eventually these are eliminated and a good product is given to the customer. It intends to find all possible defects and/or identify risks which final user may face in real life while using, the software. It works on the principle that no software is defect free, but less risky software is better and more acceptable to users. The tester's job is to find out defects so that they will be eventually fixed by developers before the product goes to a customer. Completion of testing must yield number of defects which can be analysed to find the weaker areas in the process of software development. No amount of testing can show that a product is defect free as nobody can test all permutations and combinations possible in the given software. Software testing is also viewed as an exercise of doing a SWOT analysis of software product where we can build the software on the basis of strengths of the process of development and testing, and overcome weakness in the processes to the maximum extent possible.

Strengths:

Some areas of software are very strong, and no (very less) defects are found during testing of such areas. The areas may be in terms of some modules, screens, and algorithms, or processes like requirement definition, designs, coding, and testing. This represents strong processes present in these areas supporting development of a good product. We can always rely on these processes and try to deploy them in other areas.

Weakness:

The areas of software where requirement compliance is on the verge of failure may represent weak areas. It may not be a failure at that moment, but it may be on the boundary condition of compliance, and if something goes wrong in production environment, it will result into defect or failure of software product. The processes in these areas represent some problems. An organisation needs to analyse such processes and define the root causes of problems leading to these possible failures. It may be attributed to some aspects in the organisation such as training, communication, etc.

Opportunity:

Some areas of the software which satisfy requirements as defined by the customer, or implied requirements but with enough space available for improving it further. This improvement can lead to customer delight (it must not surprise the customer). These improvements represent ability of the developing organisation to help the customer and give competitive advantage. It decides the capability of the developing organisation to provide expert advice and help to the customer for doing something better.

Threats:

Threats are the problems or defects with the software which result into failures. They represent the problems associated with some processes in the organisation such as requirement clarity, knowledge base and expertise. An organisation must invest in making these processes stronger. Threats clearly indicate the failure of an application, and eventually may lead to customer dissatisfaction.

3.13 WORKBENCH

Workbench is a term derived from the engineering set-up of mass production. Every workbench has a distinct identity as it takes part in the entire development life cycle. It receives something as an input from previous workbench, and gives output to the next workbench. This can be viewed as a huge conveyor belt where people are working their part while the belt is moving forward. The complete production and testing process is defined as set of interrelated activities where input of one is obtained from the output of previous activity, and output of that activity acts as an input to the next. Each activity represents a workbench. A workbench comprises some procedures defined for doing a work, and some procedures defined to check the outcome of the work done. The work may be anything during software development life cycle such as collecting the requirements, making designs, coding, testing, etc. Organisational process database refers to the methods, procedures, processes, standards, and guidelines to be followed for doing work in the workbench as well as for checking whether the processes are effective and capable of satisfying what customer is looking for in the outcome. There are standards and tools available for doing the work and checking the work in the work bench. While checking/testing a work product in a workbench, if one finds deviations between expected result and actual result, it may be considered as defect, and the work product and the process used lot development needs to be reworked.

3.13.1 Tester's Workbench:

Tester's workbench is made of testing process, standards, guidelines and tools used for conducting tests and checking whether the test processes applied are effective or not. For every workbench, there should be a definition of entry criteria, process of doing/checking the work, and exit criteria. For testers, there must be a definition of all things that enter the tester’s workbench. These may be defined in a test plan. Let us discuss with an example of test-case execution as one activity represented by a workbench.

Examples of Tester's Workbench:

Considering a typical system testing life cycle for a product/project, the different work benches tor a tester may be defined as follows. Kindly note that it is not an exhaustive list but a representative one. As one goes into finer details, there may be many more workbenches in each of these defined below.

- Workbench for creating test strategy
- Workbench for creating a test plan
- Work bench for writing test scenario
- Workbench for writing test cases
- Workbench for test execution
- Workbench for defect management
- Workbench for retesting
- Workbench for regression testing

The following is a typical workbench described for system testing execution.

Inputs to Tester's Workbench:

Inputs may be test scenario, test cases, and work products, documentation associated with a work product, test environment, or test plan depending upon the location of workbench in life cycle. The software work product is delivered to the tester as described in delivery note supplied by development team. Delivery note must contain any known issue which tester needs to know before performing testing.

Do Process:

The software undergoes testing as per defined test case and test procedure. This may be guided by organisational process database defining testing process. 'Do process' must guide the normal tester while doing the process.

Check Process:

Evaluation of testing process to compare the achievements as defined in test objectives is done by ‘check processes. Check process helps in finding whether ‘do processes’ have worked correctly or not.

Output:

Output must be available as required in form of test report and test log from the test process. Output of the tester's workbench needs to have an exit criteria definition.

Standards and Tools:

During testing, the tester may have to use several standard and tools. Standards may include how to install the application, which steps are to be followed while doing testing, how to capture defects, etc. There may be several tools used in testing like defect management tools, configuration management tools, regression testing tools, etc.

Remark:

If ‘check processes’ find that ‘do processes’ are no able to achieve the objectives defined for them, it must follow route of rework. This is a rework of ‘do process’ and not of work product under testing. This ensures that all incapable processes are captured so that these can be taken for improvement.

There may be two more criteria in the work bench, viz. suspension criteria for the work bench resumption criteria for the workbench guided by organisational policies and standards. Suspension criteria define when the testing process needs to be suspended or halted, whereas resumption criteria defines when it can be restarted after such halt or suspension. If there are some major problems in inputs or standards and tools required by the workbench, ‘do/check processes may be suspended. When such problems are resolved, the processes may be restarted.

Testing process may be defined as a process used to verify and validate that the system structurally and functionally behaves correctly as defined by expected result. Components of testing process may include the following.

- Giving inputs (program code) to tester from previous work bench
- Performing work (execute testing) using tools and standards
- Following a process of doing and checking whether test process is capable or not
- Produce output (test results and test log) which may act as an input to the next work bench

Check process must work to ensure that results meet specifications and standards, and also that the test process is followed correctly. If no problem is found in the test process, one can release the output in terms oftest results. If problems are found in test process, it may need rework. Fig 3.2 shows a schematic diagram of a workbench.

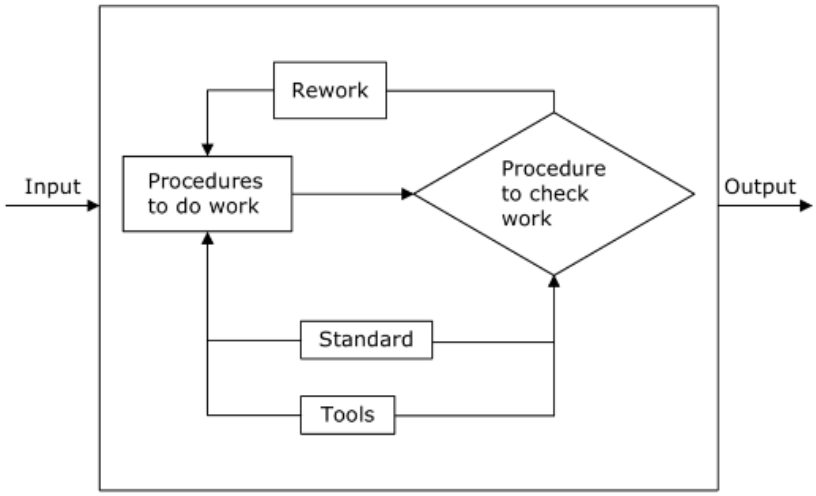


Fig. 3.2 Typical workbench

3.14 IMPORTANT FEATURES OF TESTING PROCESS

Testing is characterised by some special features, as given below.

Testing is a Destructive Process, but it is Constructive Destruction:

Testing involves a systematic destruction of a product with the intent to find the defects so that these would be fixed before the product is given to the customer. While executing tests, a tester goes about testing an application using some valid/invalid inputs to find the response of software to each of these conditions. The ability of the tester to break the application can make him successful. For devising negative testing, one needs to build the scenario with destructive mentality.

Testing Needs a Sadistic Approach with a Consideration That There is a Defect:

A tester cannot certify that a work product is defect free. He needs to go through the software work product and hunt for defects. If the defect is not found, it directly means that there is some problem with testing process. Defect-free product does not exist. Testers are expected to identify the risks to the final users and test the product accordingly.

If the Test Does Not Detect a Defect Present in the System, it is an Unsuccessful Test:

Success of testing lies in its ability to find defects or threats and weaker areas of software work product and the processes supporting these areas. Testers who cannot find defects are unsuccessful testers. Testing is considered as an investment only when it reduces the probability that software may fail at customer end.

A Test That Detects a Defect is a Valuable Investment for Development As well As Customer, it helps in Improving a Product

The root cause analysis of defects can show where the application and process can be and needs to be improved. It also helps in identifying the weaker areas of the processes used for developing software. Weaker areas are analysed to find the process lacunae and take actions on these areas and strengthen them. Thus, defect finding helps in improving processes which can result in increasing customer satisfaction. Testing with an intention to find and fix the problems in processes can be considered as an investment by customer/organisation.

Some organisations use final black box testing as an acceptance testing for the application. If no defect is found in system testing, the software is delivered to the customer. If the sole purpose of testing is to validate specifications implemented then,

- Testing is an unnecessary/unproductive activity as it does not consider invalid scenarios. No amount of testing can certify that there is no defect in the product. There must be a probability of finding the defect in testing.

- Testing is designed to compensate for ineffective software development process, it cannot give results. The defect indicates a process deficiency and it must be fixed by improving the processes. Testing cannot improve the quality of product. Defect is a symptom of something failing and one must try to fix the root cause and not only the symptom.

- If development methodology designed and implemented by a team is not correct, testing cannot compensate for it. Testing is not meant to certify the work products but to find the defects.

- Testing is a separate discipline and may get affected by as well as affects software development processes. Better processes of development and testing can reduce the chances of failure of product at customer place and subsequent customer complaints.

It is Risky to Develop Software and Not to Test it Before Delivery:

Not providing sufficient resources, time and support for testing activities is a common scenario across the industry. There is a belief that less-tested software means less defects, less rework, less scrap, and less-corrective actions which means higher profits. But this may result into customer dissatisfaction as the detects will get exposed at customer site. Reducing the coverage of testing is another risk associated with software. Software testing must give a desired level of confidence to users that system will not fail. Less testing reduces this confidence level and increases a probability of failure at customer site.

With High Pressure to Deliver Software as Quickly as Possible, Test Process Must Provide Maximum Value in Shortest Timeframe:

This approach is adopted in test strategy designing where the efficiency and effectiveness of testing is defined. Generally, test cases are categorised into installation testing, smoke testing, and sanity testing before going into further detailed testing. Test cases which represent scenarios that may occur with higher probability can help in reducing probabilities of failure in production environment. An organisation may define such test cases with probability aspect such as high, medium, and low or allocate the numbers indicating priority of execution. Probability of occurrence of adefect may not have any relationship with severity as severity talks about type of failures.

Testing is no longer an after-programming evaluation to certify that the software works, but supposed to indicate the confidence level that product will work at customer site. Testing can give the SWOT analysis of development process. Thus, testing is adjunct to software development life cycle.

Testing starts at the proposal level and ends only when software application is finally accepted by user customer. Every stage of development and every work product must go through stages of review and formal approval to ensure that it can go to the next stage. But it is a

key to ensure quality at each work product and each phase of software development life cycle.

Highest Payback Comes from Detecting Defect Early in Software Development Life Cycle and Preventing Defect Leakage/Defect Migration from One Phase to Another:

Defects are the problems or something wrong happening in software as well as the development process used for making software. Every defect indicates failure of the process at some place or another. It is always economical to fix the defects as and when they appear, and conduct an analysis to find the root causes of the defects rather than waiting till it hits the software product and user, again and again. It is always beneficial to do a root cause analysis and fix the problem areas at the earliest possible time. The investment in testing can be worthwhile only if it is capable of finding defects as soon as it is introduced in the work product and also can prevent any potential defect from getting introduced. Defect, if nor corrected in the phase where it is introduced, leaks to the next stage and creates a larger problem. This is termed 'phase contamination'. Defect keeps on migrating from one phase to next phase, till it hits back the users at some point of time.

Organisation's Aim Must Be Defect Prevention Rather That Finding and Fixing a Defect:

The major misconception about testing is that it is considered as a fault-finding mission. Instead, it must be viewed as a defect-prevention mission to avoid critical problems in software development process by initiating actions to prevent any recurrence of problems. Finding and fixing the problem is not a good approach as the basic cause of defect is never addressed. It needs analysis of root causes, and defect prevention mechanism-that is installed and operational-to prevent recurrence as well as removal of potential problems.

3.15 MISCONCEPTIONS ABOUT TESTING

At many places, software testing is termed 'quality assurance (QA)' activity. In reality testing is a quality control (QC) activity. There are many other misconceptions about software testing, as listed below.

Anyone Can Do Testing, and No Special Skills Are Required for Testing:

Many organisations have an approach that anyone can be put in testing. They give the task of testing to developers on the bench or people asking for 'light duty'. Many people involved in testing do not have any experience in testing or in the domain in which testing is done. Test planning, test case writing, and test data definition using different methodologies may not be possible with unskilled people. If the organisation considers investment in special skills as a waste of time and money, it may result in disaster for customer/user.

Testers Can Test Quality of Product at the End of Development Process:

This is a typical approach where system testing or acceptance testing is considered as qualification testing for software. Few test cases out of infinite set of possibilities are used for certifying whether the software application works or not. The customer may be dissatisfied as the application does not perform well as per his expectations. Sometimes, the defects remain hidden for an entire life cycle of software without anybody knowing that there was a defect.

Defects Found in Testing Are Blamed on Developers:

Another common misconception regarding defects found in testing is blaming developer for defects. Though two-third of defects are due to wrong requirements, yet developers are mostly blamed for detects in software development. Also, some surveys indicate that most of the defects can be attributed to faulty development processes. Developers are responsible for converting the design into code by using the standards or guidelines available. Ensuring good inputs to developer's workbench is a responsibility of the management.

Defects Found by Customer Are Blamed on Tester:

Testers perform testing by executing few list cases and try to cover some part of software program to check whether the program performs as intended or not. No one can say that testing can ensure 100% coverage. If no defect is found during testing, it does not indicate that the software program is defect free. There may be few defects left in the product which can be found only in real life. One must do a root cause analysis of the defects found, and try to learn from the experiences to ensure that a better product is produced and similar defects do not recur next time. But no one can blame testers for defects reported by customer.

3.16 PRINCIPLES OF SOFTWARE TESTING

Testing needs to be performed according to processes defined for it. It needs skilled and trained people to break the application and demonstrate the problems or defects in the software product. Some key points in software testing are as follows.

Programmers/Team Must Avoid Testing Their Own Work Products:

Everybody is in love with the work product he/she has made. Also, the approach of an individual remains the same and hence, approach-related defects cannot be found in self-review or self-testing. A second opinion is essential which can add value to a work product. Though self-review is a good to for retrospection, yet it has many limitations.

Thoroughly inspect Results of Each Test to Find Potential Improvements:

Test results show possibilities of weaker areas in the work product and the problems associated with the processes used for developing a work product. The defects found in test log do not form an exclusive list of all problems with the application, but indicate the areas where development team and management must perform a root cause analysis. Corrective actions are to be planned and executed to prevent any possible recurrence of similar defect and make software better. Defects indicate process failures.

Initiate Actions for Correction, Corrective Action and Preventive Actions:

Defect identification, fixing and initiation of action to prevent further problems are the natural ways of making better products and improve processes. Corrections of the defect are done by the developers. But one must ensure that corrective and preventive actions are initiated for making better products again and again.

Establishing that a program does what it is supposed to do, is not even half of the battle and rather easier one than establishing that program does not do what it is not supposed to do. This is negative testing driven by risk assessment for the final users. Roughly, testing may involve 5% positive testing and 95% negative testing.

3.17 SALIENT FEATURES OF GOOD TESTING

Defects indicate the quality of software under testing, development and test processes used for making it. Testing is a life-cycle activity where the testers take part in testing right from proposal stage till the applicationis finally accepted by the customer/user. Good software testing involves testing of the following.

Capturing User Requirements:

The requirements defined by the users or customer as well as some implied requirements (which are intended by the users but not put in words) represent the foundation on which software is built. Intended requirements are to be analysed and documented by testers so that they can write the test scenario and test cases for these requirements. User requirements involve technical, economical, legal, operational, and system requirements. Generally, a user is able to specify only functional and nor functional requirements which form a part of operational requirements and legal requirements but definition of other requirements is a responsibility of development organisation.

Capturing User Needs:

User needs may be different from user requirements specified in software requirement specifications. User needs may include present and future

requirements and other requirements which may include process requirements (including definition of deliverables) and implied requirements. Elicitationofrequirements is to be done by the development organisation to understand and interpret the requirements.

Design Objectives:

Design objectives state why a particular approach has been selected for building software. The selection process indicates the reasons and criteria framework used for development and testing. How an applications functional requirements, user interface requirements, performance requirements be defined in an approach document.

User Interfaces:

User interfaces are the ways in which the user interacts with the system. This includes screens and other ways of communication with the system as well as displays and reports generated by the system. User interfaces should be simple, so that the user can understand what he is supposed to do and what the system is doing. Users' ability to interact with the system, receive error messages, and act according to instructions given is defined in the user interfaces.

Internal Structures:

Internal structures are mainly guided by software designs and guidelines or standards used for designing and development. Internal structures may be defined by development organisation or sometimes defined by customer. It may talk about reusability, nesting, etc. to analyse the software product as per standards or guidelines. It may include commenting standards to be used for better maintenance. Every approach may have some advantages/disadvantages, and one needs to weigh the benefits and costs associated with them to get a better solution.

Execution of Code:

Testing is execution of a work product to ensure that it works as intended by customer or user, and is prevented from any probable misuse or risk of failure. Execution can only prove that application, module, and program work correctly as defined in requirement and interpreted in design. Negative testing shows that application does not do anything which is detrimental to the usage of a software product.

3.18 TEST POLICY

Test policy is generally defined by the senior management covering all aspects of testing. It decides the framework of testing and its status in overall mission of achieving customer satisfaction. For project organisations, test policy may be defined by the client while for product organisation, it is decided by senior management.

3.19 TEST STRATEGY OR TEST APPROACH

Test strategy defines the action part of test policy. It defines the ways and means to achieve the test policy. Generally, there is a single test policy at organisation level for product organisations while test strategy may differ from product to product, customer to customer and time to time. Some of the examples of test strategy may be as follows.

- Definition of coverage like requirement coverage or functional coverage or feature coverage defined for particular product, project and customer.
- Level of testing, starting from requirements and going up to acceptance phases of the product.
- How much testing would be done manually and what can be automated?
- Number of developers to testers.

3.20 TEST PLANNING

Test planning is the first activity of test team. If one does not plan for testing, then he/she is planning for failure. Test plans are intended to plan for testing throughout software development life cycle. Test plans are defined in the framework created by test strategy and established by test policy. Test plans are made for execution which involves various stages of software testing associated with software development life cycle. Test plan should be realistic and talk about the limitations and constraints of testing. It should talk about the risks and assumptions done during testing.

Plan Testing Efforts Adequately with an Assumption That Defects Are There

All software products have defects. Test planning should know the number of defects it is intending to find by executing the given test plan. Test plan should cover the number of iterations required for software testing to give adequate confidence required by customer (to show that software will be usable). Defect found in testing is an investment in terms of process improvement opportunity. Test plan is successful if intended number of defects is found.

Defects Are Not Found, it is Failure of Testing Activity:

There are many defects in software. If no (less) defects are found in testing, then it does not mean that there are no (less) defective in the product. It may mean that the test cases are not complete or adequate, or the test data is not effective in locating defects in the software product. Testing is intended to find defects. If defects are not found, the testing process may be considered as defective. Every defect found is an investment as it reduces a probability of any defect which customer may find. If more number of defects is found, it means the development process is problematic.

Successful Tester is Not One Who Appreciates Development but One Who Finds Defects in the Product:

Success of testing is in finding a defect and not certifying that application or development process is good. Successful testers can find more defects with higher probabilities of occurrences and higher severities of failure. Tester should find defects, which have a probability of affecting common users and thus contribute to a successful application.

Testing is Not a Formality to be completed at the End of Development Cycle:

Testing is not a certifying process. It is a life-cycle activity and should not be the last part of a development life cycle before giving the application to customer/user. Acceptance testing and system testing are the integral parts of software development where the certification of application is done by the customer but complete test cycle is much more than black box testing.

Software testing includes the following.

- Verification or checking whether a right process is followed or not during development life cycle.
- Validation or checking whether a right product is made or not as per customer's need.

Some differences between verification and validation are shown in Table 3.4.

Table 3.4 Difference between Verification and Validation

Verification	Validation
Verification is an activity where we check the work products with reference to standards, guidelines, and procedures.	Validation is an activity to find whether the software achieves whatever is defined by requirements.
Verification is prevention based. It tries to check the process adherence.	Validation is detection based. It checks the product attributes.
Verification talks about a process, standard and guidelines.	Validation talks about the product.
Verification is also termed 'white box testing' or 'static testing' as the work product undergoes a review.	Validation is also termed 'black box testing' or 'dynamic testing' as work product is executed.
Verification may be based on opinion of reviewer and may change from person to person.	Validation is based on facts and is generally independent of a person.
Verification can find about 60% of the defects.	Validation can find about 30% of the defects.
Verification involves the following. • reviews • walkthroughs • inspection • audits	Validation involves all kinds of testing. • system testing • user interface testing • stress testing
Verification can give the following. • statement coverage • decision coverage • path coverage	Validation can give the following. • requirement coverage • feature coverage • functionality coverage
Some parts of software can undergo verification only, as given below. • coding guidelines • nesting • commenting • declaration of variables	Some characteristics of software can be proven by validation only, as given below. • stress testing • performance testing • volume testing • security testing

3.21 TESTING PROCESS AND NUMBER OF DEFECTS FOUND IN TESTING

Testing is intended to find more number of defects. Generally, it is believed that there are fixed number of defects in a product and as testing finds more defects, chances of the customer finding the defect will reduce. Actually, the scenario is reverse. As we find more and more defects in a product, there is a probability of finding some more defects. This is based on the principle that every application has defects and every test team has some efficiency of finding defects. It is governed by the test team's defect-finding ability. Let us say the organisational statistics shows that after considerable testing and use of application by a user, number of defects found is three per KLOC; test planning must intend to find three defects per KLOC for the program under testing. The number of defects found after considerable testing will always indicate possibilities of existing number of defects. Fig 3.3 shows a relationship between number of defects found and probability of finding more defects.

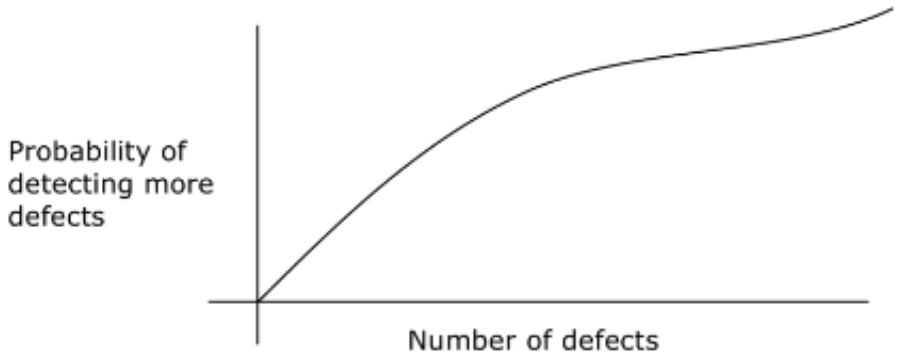


Fig. 3.3 Defect trend

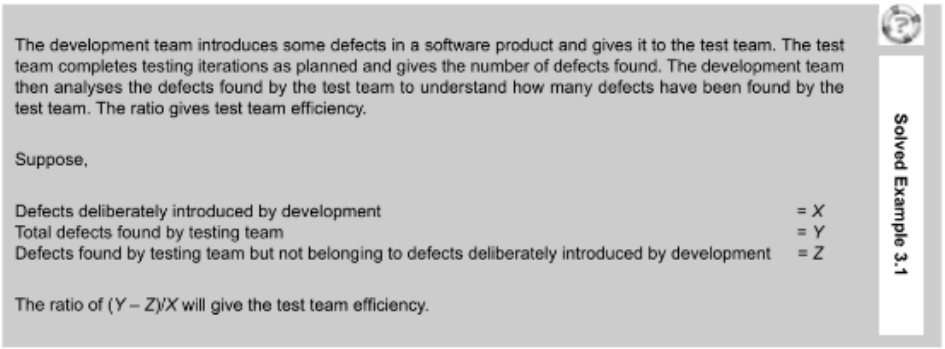
3.22 TEST TEAM EFFICIENCY

Test team efficiency is a very important aspect for development team and management. If test team is very efficient in finding defects, less iterations of testing are required. On the other hand, if development team is less efficient in fixing defects, more iteration of resting and defect fixing may be required.

The test team has some level of efficiency of finding defects. Suppose the application has 100 defects and the test team has an efficiency of 90%, then it will be able to find 90 defects. Thus, if the test team finds 180 defects (considering some efficiency), it means that there are 200 defects in the software product.

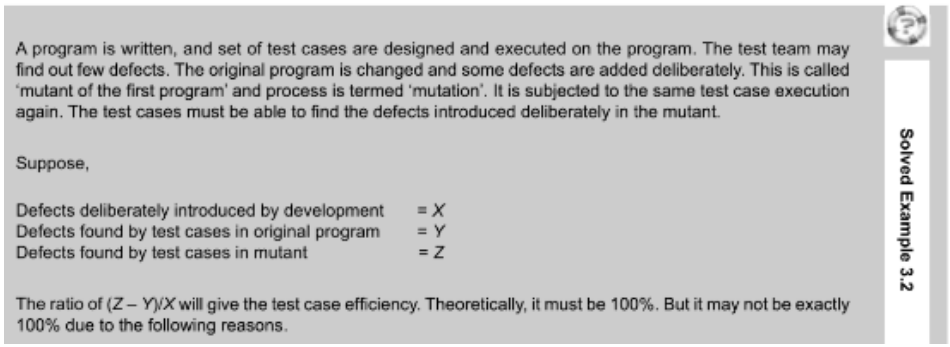
Every test manager must be aware of the efficiency of a test team that he/she is working with. Often, test managers and project managers try to access the test team efficiency at some frequency. The process may be as defined below.

Ideally, the test team efficiency must be 100% but in reality, it may not be possible to have test teams with efficiency of 100%. It must be very close to 100% in order to represent a good test team. As it deviates away from 100%, the test team becomes more and more unreliable. Test team efficiency is dependent on organisation culture and may not be improved easily unless organisation makes some deliberate efforts.



3.23 MUTATION TESTING

Mutation testing is used to check the capability of test program and test cases to find defects. Test cases are designed and executed to find defects. If test cases are not capable of finding defects, it is a loss for an organisation (as it requires time to write and execute test cases). This is also termed "test case efficiency'.



3.23.1 Reasons for Deviation of Test Team Efficiency From 100% for Test Team As Well As Mutation Analysis:

Though desirable, it is very difficult to get a test team with 100% efficiency of finding defects and test cases with 100% efficiency of finding defects. Some of the reasons for deviation are listed below.

Camouflage Effect:

It may be possible that one defect may camouflage another defect, and the tester may not be able to see that defect, or test case may not be able to locate the hidden defect. It is called ‘camouflage effect’ or ‘compensating defects’ as two defects compensate each other. Thus, defect introduced by developer may not be seen by the tester while executing a test case.

Cascading Effect:

It may be possible that due to existence of a certain defect, few more defects are introduced or seen by the tester. Though there is no problem in the modification, detects are seen due to cascading effect of one defect. Thus, defects not introduced by developer may be seen by tester while executing a test case.

Coverage Effect:

It is understood that moody can test 100%, and there may be few lines of code or few combinations which are not tested at all dive to some reasons. If defect is introduced in such a part which is not executed by given set of test cases, then tester may not be able to find the defect.

Redundant Code:

There may be parts of code, which may not get executed under any condition, as the conditions may be impossible to occur, or some other conditions may take precedence over it. If developer introduces a defect in such parts, testers will not be able to find the defect as that part of code will never get executed.

3.24 SUMMARY

This chapter establishes the basics of software testing. It starts with a historical perspective of testing and then explains how testing evolved from mere debugging to defect prevention technique. It then discusses the benefits of independent testing. 'TQM' concept of testing, 'Big Bang' approach of testing, and benefits of 'TQM' testing are elucidated in detail.

3.25 EXERCISE

1) Explain the evolution of software testing from debugging to prevention based testing.
2) Explain why independent testing is required.
3) Explain big bang approach of software testing,
4) Explain total quality management approach of software testing.
5) Explain concept of TQM cost perspective.
6) Explain testing as a process of software certification.
7) Explain the basic principles on which testing is based.

3.26 REFERENCES

1. Software Testing and Continuous Quality Improvement by William E. Lewis CRC Press Third 2016.

2. Software Testing: Principles, Techniques and Tools M. G. Limaye TMH 2017.

3. Foundations of Software Testing Dorothy Graham, Erik van Veenendaal, Isabel Evans, and Rex Black Cengage Learning 3 rd.

4. Software Testing: A Craftsman‟s Approach Paul C. Jorgenson CRC Press 4 th 2017.

5. https://www.techtarget.com/whatis/definition/software-testing

6. https://www.softwaretestinghelp.com/types-of-software-testing/

7. https://prepinsta.com/software-engineering/big-bang-approach/

*****

4

CHALLENGES IN SOFTWARE TESTING

Unit Structure

4.0 OBJECTIVES

After studying this chapter the learner would be able to:

- Understand different challenges in testing
- Understand different approaches to testing.
- Understand different types of defects.
- Differentiate between “White Box testing”, “Black Box testing” &“Grey Box testing”

4.1 CHALLENGES IN TESTING

Testing is a challenging job. Challenges in testing are different on different fronts. On one front, it needs to tackle with problems associated with development team. On second front, it has customers to tackle with. Management may have problems with understanding testing approach and may consider it as an obstacle to be crossed before delivering the product to the customer. There may be problems related to testing process as well as development process. Major challenges faced by test teams are as follows.

- Requirements are not clear, complete, consistent, measurable and testable. These may create some problems in defining test scenario and test cases. Sometimes, a configuration management issue is faced when the development team makes changes in requirements but test team is not aware of these changes.

- Requirements may be wrongly documented and interpreted by business analyst and system analyst. These knowledgeable people are supposed to gather requirements of customers by understanding their business workflow. But sometimes, they are prejudiced based on their earlier experiences.

- Code logic may be difficult to capture. Often, testers are not able to understand the code due to lack of technical knowledge. On the other hand, sometimes, testers do not have access to code files.

- Error handling may be difficult to capture. There are many combinations of errors, and various error messages and controls are required such as detective controls, corrective controls, suggestive controls, and preventive controls.

4.1.1 Other Challenges In Testing:

More bugs found in software introduce additional iterations of fixing defects, retesting, and regression testing of a product. It means more efforts, delayed shipment to customer and late payment receipt by organisation. Often, testers are blamed for delays in delivery.

- Badly written code introduces many defects. Code may not be readable, maintainable, optimisable, and may create problems in future. Defects may not be fixed correctly, and fixing of defects may introduce more defects called 'regression defects'.

- Bad architecture of software cannot implement good requirement statement. What developers do in reality is that they implement the design and not the requirements. Bad architecture creates complex code and adds many defects to the software product.

- Testing is considered as a negative activity. Often, testers need to reject builds if problems are found in it which does not satisfy exit criteria. It is a difficult situation as organisational fund flow may be depending upon successful delivery of system, and it gets affected due to such rejection.

- Testers find themselves in lose-lose situation in resting. If more defects are found, application delivery is delayed and testers are blamed for such delay. On the other hand, if testing is not done properly, customer complaints are possible and again testers are held responsible for it.

4.2 TEST TEAM APPROACH

Type of the organisation and type of the product being developed define a test team. There may or may not be a separate team doing testing if management does not recognise its importance, or the application under development demands this scenario. There are four approaches of software testing team.

4.2.1 Location of Test Teams In An Organisation:

Generally, test team is located in an organisation as per testing policy. It may vary from organisation to organisation, project to project and customer to customer. Following are some of the approaches used for locating test team organisationally.

Independent Test Team:

Independent test team may not be reporting to development group at all, and are independent of development activities. They may be reporting independently to senior management or customer. Presence of test manager is essential to lead the test team. Such test teams may be shown as in figure 4.1.

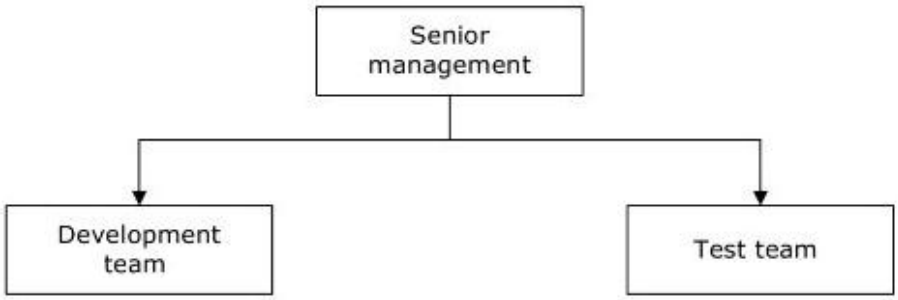


Fig. 4.1 Organisational structure of test team independent of development team

Advantages of Independent Test Team:

- The test team is not under delivery pressure. They can take sufficient time to execute complete testing as per definitions of iterations, coverage, etc
- Test team is not under pressure of 'not finding' a defect. They are considered as the certifiers of a product and must be able to find every conceivable fault in the product before delivery.
- Independent view about a product is obtained as thought process of developers and testers may be completely different.
- Expert guidance and mentoring required by test team for doing effective testing may be available in the form of a test manager.

Disadvantages of Independent Test Team:

- There is always ‘us’ vs ‘them’ mentally between development team and test team. Team synergy can be lost as developers take pride in what they develop while testers try to break the system.
- Testers may not get a good understanding of development process as development team tries to hide the process lacunae from them. Testers are treated as outsiders.
- Sometimes, management may be inclined excessively towards development team or test team, and the other team may feel that they have no value in an organisation.

Test Team Reporting to Development Manager:

If the test team is reporting to development manager, then they can be involved from the start of project till the project is finally closed. Such test team may be shown in Fig 4.2.

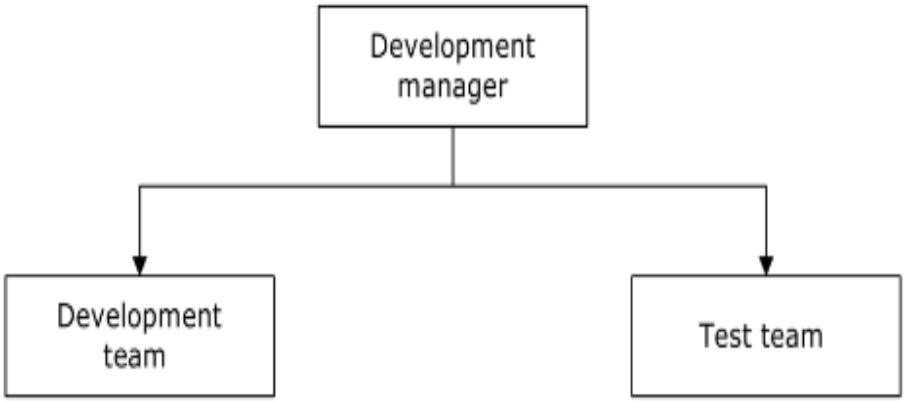


Fig 4.2 Organisational Structure of test team reporting to development manage

Advantages of Test Team Reporting to Development Manager:

- There is a better cooperation between development team and test team as both are part of the same team.
- Test team can be involved in development and verification/validation activities from the start of the project. It gives them good understanding of requirements.
- Testers may get a good understanding of development process and can help in process improvement.

Disadvantages of Test Team Reporting to Development Manager:

- Expert advice in the form of test manager may not be available to testers. In case testers need some guidance or mentoring, they may have to rely on an external person.
- Sometimes, development managers are more inclined towards development team. Defects found by test team are considered as hurdles in delivery process.
- Often, testers start perceiving the product from developer's angle and their defect finding ability reduces.

Matrix Organisation:

In case of matrix organisation, an effort is made to achieve the advantages of both approaches, and get rid of disadvantages of both approaches. Test team may be reporting functionally to the development manager while administratively it reports to the test manager.

4.2.2 Developers Becoming Testers:

Sometimes, those who work as developers in initial stages of development life cycle take the role of testers when the latter stages of life cycle are executed. This is a common practice while working with simple software

which does not need very structured testing. Developers becoming testers can be suitable when the application is technologically heavy.

Advantages of This Approach:

- Developers do not need another knowledge transfer while working as a tester. The knowledge transfer that they received at initial stages of development can be used by them in both roles.
- Developers have better understanding of detail design and coding, and can test the application easily.
- For automation, some amount of development skill is required in writing the automation scripts. Developer can adapt themselves in automation testing better as they have the ability to create code.
- It is less costly as there is no separate test team. It provides staff balancing to some extent. Initially, the development team is large but as SDLC comes to an end, the test team becomes larger than the development team.
- Psychological acceptance of defects is not a major issue as developers themselves find the defects.

Disadvantages of This Approach:

- Developers may not find value in performing testing. They may put more time in developing/optimising code than executing serious testing.
- There may be blindfolds while understanding requirements or selection of approach and developer may not be willing to find more defects. Understanding or approach related defects may not be found. Platform or database related defects may not be uncovered as developers may feel that as technological limitations. As per developers technology is creating the problem.
- Developers may concentrate more on development activities, which is their primary responsibility and may neglect testing activities.
- Development needs more of a creation skill while testing needs more of a destruction skill. It is difficult to have a team having both skills at a time.

4.2.3 Independent Testing Team:

An organisation may create a separate testing team with independent responsibility of testing. The team would have people having sufficient knowledge and ability to test the software.

Advantages of This Approach:

- Separate test team is supposed to concentrate more on test planning, test strategies and approach creating test artifacts, etc.

- There is independent view about the work products derived from requirement statement.
- Special skills required for doing special tests may be available in such independent teams.
- 'Testers working for customer' can be seen in such environment.

Disadvantages of This Approach:

- Separate team means additional cost for an organisation.
- Test team needs ramping up and knowledge transfer, similar to a development team.
- An organisation may have to check for rivalries between development team and test team.

4.2.4 Domain Experts Doing Software Testing:

An organisation may employ domain experts for doing testing. Generally, this approach is very successful in system testing and acceptance testing where domain specific testing is required. Domain experts may use their expertise on the subject matter for performing such type of testing.

Advantages of This Approach:

- Fitness for use' can be tested in this approach where actual user's perspective may be obtained. Domain experts will be testing software from user's perspective.
- Domain experts may provide facilitation to developers about defects and customer expectations, and maybe able to interpret requirements in the correct context.
- Domain experts understand the scenario faced by actual users and hence, their testing is realistic.

Disadvantages of This Approach

- Domain experts may have prejudices about the domain which may reflect in testing. Domain experts may have knowledge about a domain but may not understand exactly what a particular customer is looking for.
- It may be very difficult to get domain experts in diverse areas, if an organisation has projects in diverse domains.
- It may mean huge cost for the organisation as these experts cost much more than normal developers/testers.

Combination of all three approaches (3.21.2, 3.21.3, 3.21.4) can be advantageous for the organisation. One has to do a cost-benefit analysis to arrive at any decision about test team formation. Highly complex user

environment and complex algorithms may make ‘domain experts doing testing’ more effective. On the contrary, if an organisation has done many projects in similar domain in past, ‘developers becoming tester’ may be recommended as developers may have sufficient knowledge about the subject.

In addition to a test team, there are many other agencies involved in software testing as per phases of software development.

Customer/User:

Customer or users generally do acceptance testing to declare formal acceptance/rejection/changes in requirements for the product. Customer perspective is most important in software acceptance. Organisations creating prototype may rely on customer approval of prototype.

Developers:

Developers do unit testing before the units are integrated. Generally, units require stubs and drivers for testing, and developers can create the same. Sometimes, integration testing is also done by developers if it needs stubs and drivers.

Tester:

Testers perform module, integration, and system testing as independent testing. They may oversee acceptance testing. Tester's view of system testing is very close to user's view while accepting software.

Information System Management:

Information system management may do testing related to security and operability of system. They would provide the specialized skills needed for this type of testing.

Senior Management/Auditors:

Senior management or auditors appointed by senior management such as Software Quality Assurance (SQA) perform redelivery audit, smoke testing, and sample testing to ensure that proper product is delivered to the customer.

4.3 PROCESS PROBLEMS FACED BY TESTING

‘Q’ organisations consider that defects in the product are due to incorrect processes, in general, it is believed that incorrect processes cause majority (about 90%) of the working problems. Defects are introduced in software due to incapable processes of development and testing. Software testing is also a process, and prone to introduce defects in the system. If the process of software testing is faulty, it gives problems in terms of defects not found during testing but found by customer, or wrong defects found which are either ‘not a defect’, ‘duplicate’, ‘cannot be reproduced’ or ‘out of

scope' type of defects. The basic constituents of processes are people, material, machines and methods.

People:

Many people are involved in software development and testing, such as customer/user specifying requirements; business analysts/system analysts documenting requirements; test managers or test leads defining test plans and test artifacts; and testers defining test scenarios, test cases, and test data. There is a possibility that at few instances some personal attributes and capabilities may create problems in development and testing. Proper skill sets such as domain knowledge and knowledge about development and testing process may not be available.

Material:

Testers need requirement documents, development standards and test standards, guidelines, and other material which add to their knowledge about a prospective system. These documents may not be available, or may not be clear and complete. Similarly, other documents which act as a framework for testing such as test plans, project plan, and organisational process documents may be faulty. Test tools and defect tracking tools may not be available. All of these may be responsible for introducing defects in the product.

Machines:

Testers try to build real-life scenarios using various machines, simulators and environmental factors. These may include computers, hardware, software, and printers. The scenarios may or may not represent real-life conditions. There may be problems induced due to wrong environmental configurations, usage of wrong tool, etc.

Methods:

Methods for doing test planning, risk analysis, defining test scenarios, test cases, and test data may not be proper. These methods undergo revisions and updating as the organisation matures.

Economics of Testing:

As one progresses in testing, more and more defects are uncovered, and probability of customer facing a problem reduces while the cost of testing goes up. The cost of customer dissatisfaction is inversely proportional to testing efforts. It means more investment in testing efforts reduces the cost of customer unhappiness. On the other hand, the cost of testing increases exponentially. If the first ten defects are found in one hour, for finding the next ten defects, it may need two hours, and so on. If we plot both curves, then at some point, the two curves intersect each other. This point shows optimum testing point. Area before this point represents an area under testing where defective product goes to customer and customer dissatisfaction cost is higher than cost of testing, while area after this point

represents an area of over testing where cost of customer dissatisfaction is less than cost of testing.

Cost of testing curve is guided by the following:

- Defect finding ability of testing team (test team efficiency)
- Defect fixing ability of development team (defect fixing efficiency)
- Defect introduction index of development team which talks about regression defects getting introduced due to fixation of some defects discovered during testing

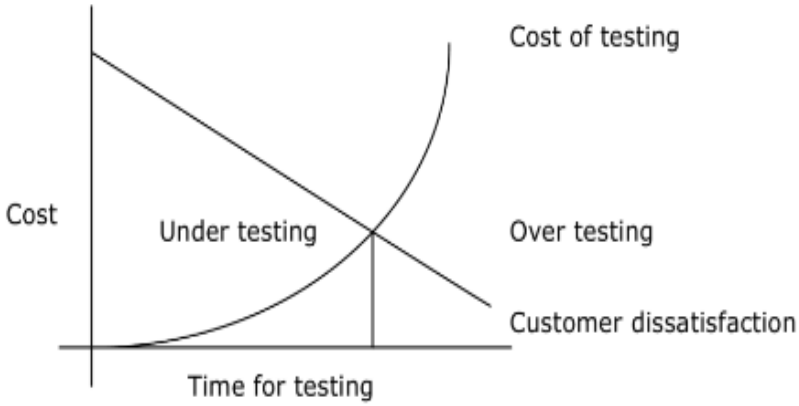


Fig 4.3 Cost of Testing and Cost of Customer Dissatisfaction

- If test team efficiency and defect fixing efficiency are 100%, and defect introduction index is zero, we need only two iterations of testing. As it goes away from ideal numbers, number of iterations increase exponentially.

Cost of customer dissatisfaction is guided by the following aspects:

- Cost of customer dissatisfaction mainly depends upon customer-supplier relationship. If an organisation has done several projects which were very successful, then the customer may not mind if some defects are there in the current project. Customer dissatisfaction curve tends to be parallel to ‘Y’ axis. On the other hand, if the customer is very finicky about defects and past performance of an organisation is not good, it may tend to be parallel to 'X' axis.
- Cost of customer dissatisfaction also depends on the type of product and its mission criticality to the customer. Customer may not like any problem in high mission-critical software while he may accept certain problems in other types of software.

4.4 COST ASPECT OF TESTING

As seen earlier, cost of quality includes cost of prevention, cost of appraisal and cost of failure. Testing may take some portion of each of these costs. It is believed that cost of quality is about 50% of the cost of

product. Testing is a costly affair and an organisation must try to reduce the cost of testing to the maximum extent possible.

There is a famous concept of efforts conversion into cost in case of software development, as effort is the major component of total cost. This may be done by standard costing method or marginal costing method as per organisation's process definition. Efforts spent by the organisation in developing and testing of application are converted at some predefined rates to arrive at the total cost of a product. Sometimes, the cost of resource varies as per the role played by a person. For example, a project manager may get more rate than developer, or an architect may get more billing than a tester.

Solved Example 3.3

Let us suppose that the project duration is 10 months with 22 days of working per month, 8 hours working per day and 100 people are working on it. Also, if conversion rate is say Rs 500 per person hour, then the cost of development and testing taken together will be as follows.

Total efforts spent on project = 10 × 22 × 8 × 100 = 176,000 hrs
Total cost = 176,000 × 500 = Rs 88,000,000

An organisation may have some additions to this cost in terms of contingencies, overheads and profit expected to arrive at sales price. If contingency is considered at 10%, overhead apportionment is considered at 10% and expected profit is considered at 20%, then

Sales price would be = 8,800,000 × 110% × 110% × 120% = Rs 127,776,000

It is a very rare scenario that a project will have 100 people working for all 10 months for the development project. Generally, development projects never have the same number of resources throughout the life cycle. Initially, it may need less number of people and as one passes through different phases of development, number of resources required increases exponentially. Once the peak activities are over, number of resources required goes down.

The same cycle is followed by testing resource requirements. As the testing phases progress, number of resources required increases exponentially. Once the peak activities are over, number of test resources goes down. Thus, for development project, costing is always dynamic.

In case of maintenance or production support type of work, number of resources remains fairly constant over a long-time horizon. Fig 4.4 indicates resources required for a development project.

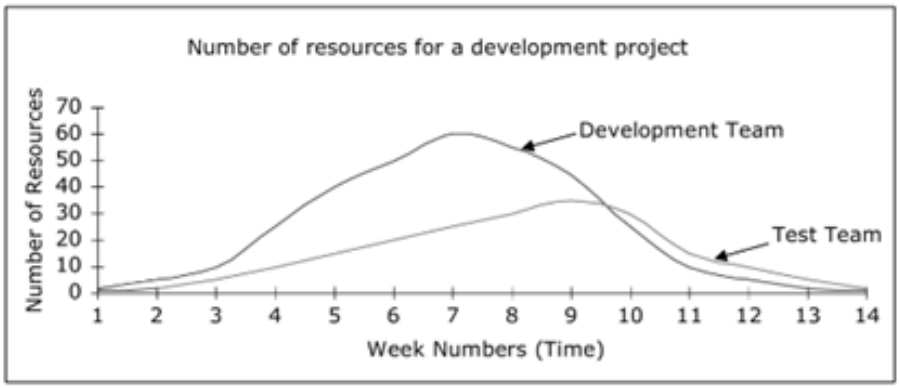


Fig 4.4 Number of resources for development project

Cost of development/manufacturing includes the cost spent in the following.

- Capturing the requirements, conducting analysis, asking queries, and elicitation of requirements
- Cost spent in designing the application including high-level designing and low-level designing
- Cost spent in writing code, integrating and creating the final product

4.4.1 Assessment of Cost of Testing:

Cost of testing may be considered as the cost of the project activities on and above normal phases of development. Various phases of development are associated with phases of verification and validation. Cost of testing is a function of two processes, viz. development process maturity and test process maturity. It has also a very close relationship with the type of application being produced, methodology of development and testing, domain, technical parameters of development, and testing.

Type of Application and Cost of Testing:

Depth and breadth of testing has direct relationship with the size and importance of an application to a user, and test efforts that the customer is ready to pay for also varies accordingly. As far as size of an application is concerned, testing follows 'Rayleigh Putnam' curve. As the size of application increases, the testing efforts increase exponentially. An organisation must evaluate is own curve on the basis of historical data, or may adapt to the existing baselines in initial phases when historical data is not available.

$$\text{Cost of testing} = k\, f(x)$$

Where 'K' is a constant and f(x) is a function of size of an application while 'x' indicates the size of the application.

An application may be defined as per its importance to the user. If user is completely dependent upon the application, he may want more confidence and thus, invest more in testing. On the other hand, if software does not have business criticality, the customer may not be ready to pay that much cost.

Development and Test Methodology:

Development and test methodologies also affect testing efforts and costs. It is generally believed that waterfall development methodology can produce robust software and testing efforts required are much lesser. Agile methodologies may need huge testing because there is more stress on regression testing, as each iteration of development is over. Iterative

development has more testing costs as there is a change in requirements, design, coding, etc.

Test methodologies have an impact on test efforts and costs. An organisation conducting phase-wise testing has lesser cost than the organisation facing testing at the last phase of development. Life cycle testing is very cheap in comparison to testing at the last phase of development. Usage of regression testing tools may increase the cost of testing in initial phases while in the long term, overall cost of testing is reduced due to automation.

Domain and Technological Aspects:

Application domain plays an important role, along with criticality of the application, to the users. If the application is going to affect human life, there may be many regulations coming into the picture. The customer will be more cautious about testing and test results. There may be more legal implications if sufficient and adequate testing is not done. Some other types of software affecting huge sum of money are also regulated by the rules, regulations and laws of the place where it is used.

It is believed that technology also plays an important role in deciding the extent of testing. Object-oriented development faces different challenges in testing compared to normal development without any object usage.

There may be more thrust on integration testing rather than unit testing. Old development languages and databases sometimes make testing difficult.

Maturity of Development and Testing Processes:

Development and testing process maturity is an important issue in deciding the cost of testing. Theoretically, testing is not at all required if development can achieve the defined output. If development process is highly matured and can achieve the expected output, then testing may be considered as wastage. Many engineering organisations have reached the phase of zero defect and 'zero' inspection. But software application development may not have achieved a level of maturity to produce defect-free products. Some amount of testing is required to find all conceivable faults so that development team can fix them.

Many organisations have a development phase followed by one complete iteration of testing. It must find all defects so that these are eventually fixed in rework phase. Second iteration of testing must achieve the required level of confidence that application will not fail.

Testing involves all three components of cost of quality mentioned below.

4.4.2 Cost of Prevention in Testing:

Cost of prevention is the cost incurred in preventing defects from entering into a system. In various phases of verification and validation, cost of prevention is distributed. Some of these are discussed below.

Cost Spent in Creation of Verification and Validation Artifacts:

This part of cost is spent when the project team and test team create various artifacts related to verification and validation at different phases of software development. Cost of planning includes creating test plans and quality plans so that quality can be appraised correctly. This phase may also include the time and effort spent in creation of guidelines for testing, reviews, creation of checklists, writing of test cases and test data along with test scenarios. In case of test automation, cost of test-script creation may be a part of prevention cost.

Cost Spent in Training:

Testers may need training on the domain under testing. They may also need training on test process and various tools used for testing. This may include organisation-level training as well as project-specific training.

Cost of prevention is calculated as follows:

- Cost spent in creating various plans related to software verification and validation. An organisation must have a baseline definition of the effort required to create plans.
- Cost spent in writing test scenarios, test cases, creating guidelines for verification and validation, and creating checklists for doing verification and validation activities. Organisation baselines must define the time and effort required for this activity.
- Training requirements may be separately assessed depending upon the competency of test teams, type of application, and level of tool usage for testing. There may not be baseline data available in this case as it may change from project to project.

4.4.3 Cost of Appraisal In Testing:

Cost of appraisal includes cost spent in actually doing verification and validation activities. Generally, first-time verification/validation is considered under cost of appraisal. Test artifacts also need reviews and testing to confirm that they are correct. Checklists needed for the reviews must also be reviewed before they are used.

Irrespective of whether time and efforts are spent by developers or testers, all efforts spent on conducting reviews, walkthroughs, inspection, unit testing, integration testing, and system testing are considered under cost of appraisal.

Cost of appraisal is calculated as follows:

- Cost required for conducting first-time verification and validation activities can be assessed from the size of an application and organisation baseline data available.

- Time and effort required to conduct the given number of test cases may be derived from the productivity numbers, and number of test cases required can be derived from the size of an application.

4.4.4 Cost of Failure In Testing:

Cost of failure in testing accounts for all retesting, regression testing, and re-reviews conducted as the defects are found in earlier iterations. Any cost spent on and above first-time verification and validation makes a cost of failure for testing. The organisation must have a target to reduce the cost of failure continuously, as this directly affects its profitability.

Cost of failure is calculated as follows:

- On the basis of historical data available in baseline studies, one may plan number of iterations required to achieve predetermined exit criteria.

- Number of test cases per iteration and number of iterations required can give the efforts required to perform retesting and regression testing.

4.5 ESTABLISHING TESTING POLICY

Good testing is a deliberate planned effort by the organisation. It does not happen on its own, but detailed planning is required. Testing efforts need to be driven by test policy, test strategy or approach, test planning, etc. Test policy is an intent of test management about how an organisation perceives testing and customer satisfaction. It should define test objectives and test deliverables.

Test strategy or approach must define what steps are required for performing an effective testing. How the test environment will be created, what tools will be used for testing, defect capturing, defect reporting, and number of test cycles required will be a part of test strategy. It must talk about the depth and breadth of testing to ensure adequate confidence levels for users.

Test objectives define what testing will be targeting to achieve. It is better to have test objectives expressed in numbers in place of qualitative definitions. Some of the test objectives may be about code coverage, scenario coverage, and requirement coverage, whereas others may define the targeted number of defects.

Testing must be planned and implemented as per plan. Test plan should contain test objectives and methods applied for defining test scenario, test cases and test data. It should also explain how the results will be declared and how retesting will be done. It is a general expectation that automation can solve all problems regarding testing process. One thing to be noted is that- automation can increase the speed and repeatability of testing but test planning cannot be done automatically. Rather, it needs involvement of people doing testing.

4.6 METHODS

Generally, methods applied for testing efforts are defined at organisational levels. They are generic in nature and hence, need customisation. They are customised into a test plan, and any tailoring required to suite a specific project may be done. Management directives establish methods applied for testing. It includes what part will be rested nor tested, and how it will be tested. Which tools will be used for testing, defect-tracking mechanism, communication methods and if there is any decision of automation, how it will be undertaken all this must be covered by these processes. Management directives are defined in test strategy.

Testing strategy may be discussed with users/customer to get their views/buy-in about testing. It may be accomplished through meetings and memorandums, User/customer must be made aware of cost of finding and fixing defects. All stakeholders for the project must be made aware that ‘zero defects’ is an impossible condition and acceptance criteria for the project must be defined well in advance (possibly at the time of contract). Methods of using data or inputs provided by a customer must be analysed for sufficiency and correctness.

4.7 STRUCTURED APPROACH TO TESTING

Testing that is concentrated to a single phase at the end of development cycle, just before deployment, is costly. Testing is a life cycle activity and must be a part of entire software development life cycle. If testing is done only in the last phase before delivery to customer, the results obtained may not be accurate and defect fixing may be very costly. Here, it cannot show development process problems and similar defects can be found again and again. Four components of wastes involved in this type of testing aregiven below.

Waste in Wrong Development:

Wrong specifications used for development or testing will result into a wrong product and wrong testing. Even if specifications are correct, it may be wrongly interpreted in design, code, documentation, etc. The defects not found during reviews or white box testing will be discovered only at the customer's end. This may lead to high customer dissatisfaction, huge rework, retesting, etc.

Waste in Testing to Detect Defects:

If testing is intended to find all defects in product, then cost of testing will be very high. Effective reviews can reduce the cost of software testing and development. If entire responsibility for software quality is left to black box testing or final system testing, then the cost of testing and cost of customer dissatisfaction may be very high.

Wastage as Wrong Specifications, Designs, Codes and Documents Must Be Replaced by Correct Specifications, Designs, Codes and Documents:

The cost of fixing defects maybe very high in the last part of testing as there are more number of phases between defect introduction phase and defect detection phase, and defect may percolate through the development phases. Correcting the specifications, designs, codes and documents, and respective retesting/regression testing is a costly process. It can lead to schedule variance, effort variance and customer dissatisfaction. One defect fix can introduce another defect in the system.

Wastage as System Must Be Retested to Ensure That the Corrections Are Correct:

For every fixing of defect, there is a possibility of some other part of software getting affected in a negative manner. One needs to test software again to ensure that fixing of software has been correct, and it has not affected the other parts in a negative manner. Regression and retesting are essential when defects are found and fixed.

4.8 CATEGORIES OF DEFECT

Software defects may be categorized under different criteria. The categories of defects must be defined in the test plan. The definition may differ from organisation to organisation, project to project and customer to customer.

4.8.1 On Basis of Requirement/Design Specification:

- Variance from product specifications as documented in requirement specifications or design specifications represents specification related defects. These defects are responsible for Producer's ‘gap’.
- Variance from user/customer expectations as business analyst/system analyst is not able to identify customer needs correctly. These variances may be in the form of implied requirements. These are responsible for ‘Users gap’.

4.8.2 Types of Defects:

- Wrongly implemented specifications are relate to the specifications as understood by developers differing significantly from what the customer wants. These may be termed ‘misinterpretation of specifications’.
- Missing specifications are the specifications that are present in requirement statements but not available in the final product. The requirements are missed, as there is no requirement tracing through product development.
- Features not supported by specifications but present in the product represent something extra. This is something added by developers though these features are not supported by specifications.

4.8.3 Root Causes of Defects:

- Wrong requirements given by user/customer can be a basic cause of defect. This is due to the inability of the customer to put the requirements in words, or specifying requirements which are not required.

- Business analyst/system analyst interprets customer needs wrongly can be another major cause of defect. This is due to the inability of business analyst/system analyst to elicit requirements.

- System design architect does not understand requirements correctly and hence, the architecture is wrong. This may be due to communication gap or inability of the architect in understanding the requirements.

- Incorrect program specifications, guidelines, and standards are used by respective people. If the organization processes are not capable, then defects are introduced in the product so produced.

- Errors in coding represent lack of developer's skills in understanding design and implementing it correctly.

- Data entry error caused by the users while using a product. This can be possible when users are not protected adequately. This indicates design problems.

- Errors in testing-false call/failure to detect an existing defect in the product. The first part introduces defects in a correct product while the second part allows defects to go to the customer.

- Mistake in error correction, where defect is introduced while correcting some identified defect.

4.9 DEFECT, ERROR, OR MISTAKE IN SOFTWARE

The problems with software work product may be put under different categories on the basis of who has found it and when it has been found (as shown in Table 4.9)

Table 4.9 Comparison of mistake, error and defect

Mistake	Error	Defect
An issue identified while reviewing own documents, or peer review may be termed 'mistake'.	An issue identified internally or in unit testing may be termed 'error'.	An issue identified in black box testing or by customer is termed 'defect'.
Very low cost of finding mistakes and can be fixed immediately.	Slightly more cost of finding an error and needs some time for fixing.	Most costly and needs longer time for fixing defects.
Most of the time, problems and resolutions are not documented properly.	Sometimes, problems and resolutions are documented, but may not be used for process improvements.	Problems and resolutions are officially documented and used for process improvements.

4.10 DEVELOPING TEST STRATEGY

Test planning includes developing a strategy about how the test team will perform testing. Some key components of testing strategy are as follows.

- Test factors required in particular phase of development
- Test phase corresponding to development phase

Process of developing test strategy goes through the following stages.

Select and Rank Test Factors for the Given Application:

The test team must identify critical success factors quality factors 'test factors for the software product under testing. Software may have some specific requirements from user's point of view. Test factors must be analysed and prioritised or ranked. Some test factors may be related to each other (either direct or inverse relationship). The trade-off decisions may be taken after consulting with customer, if possible, when the relationship is inversed.

Identify System Development Phases and Related Test Factors:

The critical success factors may have varying importance as per development life cycle phases. One needs to consider the importance of these factors as per the life cycle phase that one is going through. The test approach will change accordingly.

Identify Associated Risks with Each Selected Test Factor in Case if it is not Achieved:

Trade-offs may lead to few risks of development and testing the software. Customer must be involved in doing trade-offs of test factors and the possible risks of not selecting proper test factor. The risks with probability and impact need to be used to arrive at the decision of trade-off.

Identify Phase in Which Risks of Not Meeting a Test Factor Need to Be Addressed:

The risks may be tackled in different ways during development life cycle phase’s counter measures for the same. As the phase is over, one needs to assess the actual impact of the risks and effectiveness of devised countermeasures for the same.

4.11 DEVELOPING TESTING METHODOLOGIES (TEST PLAN)

Developing test tactics is the job of project-level test manager/test lead. Different projects may need different tactics as per type of product customer. Designing and defining of test methodology may take the following route.

4.11.1acquire and Study Test Strategy As Defined Earlier:

Test strategy is developed by a test team familiar with business risks associated with software usage. Testing must address the critical success factors for the project and the risk involved in not achieving it.

4.11.2 Determine the Type of Development Project Being Executed:

Development projects may be categorised differently by different organisation, as shown below.

- Traditionally developed software by following known methods of development such as waterfall development life cycle. An organisation has a history available, and the lessons learned in previous projects can be used to avoid contingencies in the given project.
- Commercially off the Shelf (COTS) purchased software which may be integrated in the given software. Requirements and designs of such software may not be available, and integration in terms of parameters passing can be a major constraint.
- Maintenance activities such as bug fixing, enhancement, porting, and reengineering will have their own challenges, such as availability of design documents, compatibilities of various technologies, and code readability.
- Agile methodology of development has small iterations of development and heavy regression testing.
- Iterative method of development has continuously changing requirements and all other artifacts must be updated accordingly.
- Spiral development, where new things are added in system again and again. Generally followed methodology in spiral development is modular design, development and testing.

4.11.3 Determine the Type Of Software System Being Made:

Type of software system defines how data processing will be performed by the software. It may involve the following.

- Determine the project scope (whether it is multivendor or multisite development). Distributed development and integration of parts developed by different vendors can be a challenging task.
- New developments including scope of development and scope of testing, when the products are enhanced from existing levels.
- Changes to existing system such as bug fixing, enhancement, reengineering, and porting.

4.11.4 Identify tactical Risks Related to development:

Risks may be introduced in software due to its nature, type of customer, type of developing organisation, development methodologies used, and skills of teams. Risk may differ from project to project.

- **Structural Risks (Refers to Methods Used to Build a Product):** If the projects are supposed to use existing libraries or designs which may need complex algorithms to be written, the structure of the software may pose the highest risk. Complex structures with interfaces in relation to many other systems can make the architecture fragile.

- **Technical Risks (Refers to Technology Used to Build and Operate the System)**: If the organisation is new to a particular technology, or the technology itself is new to the world, then it can lead to this king of risk. Unproven technology or inability to work with the latest technology is the problems faced by developers as well as users.

- **Size Risks (Refers to Size of All Aspects of Software):** As the software size increases. It becomes more complex. Maintaining integrity of very big software itself is a challenge.

4.11.5 Determine When Testing Must Occur During Life Cycle:

- Testing phases starting from proposal, contract or requirement testing till acceptance testing and their integration decide the test strategy for the project.

- Previously collected information, if available, is to be used to decide how much testing is to be done, at what time and in which phases. It defines the cost of testing, cost of customer dissatisfaction and any trade-offs.

- Build tactical test plan which will be used by the test team in execution of testing related activities.

- To describe software being tested, test objectives, risks, business functions to be tested, and any specific tests to be performed.

4.11.6 Steps to Develop Customised Test Strategy:

- Select and rank quality factors/test factors as expected by the customer in the final product. Quality factors must be prioritised. Generally, the scale used is 1-99, where ‘1’ indicates higher priority while '99' indicates lower priority. No two quality factors have the same ratings.

- Identify the system development phase where these factors must be controlled. As the defects may originate in different phases, quality factors may change their priority during each phase of development life cycle.

- Identify business risks associated with system under development. If quality factors are not met, these may induce some risk in a product.
- Place risks in a matrix so that one may be able to analyse them. This may be used to devise preventive, corrective and detective measures to control risks. Table 4.10 shows how test strategy matrix can be developed.

Table 4.10 Typical test strategy matrix

Quality factors	Test phases	Require-ment	Design	Coding	Testing	Deploy-ment	Mainte-nance
Factors	Compliance Accuracy Reliability		Risks associated at different phases for different factors.				

4.11.7 Type of Development Methodology Impact Test Plan Decisions:

Table 4.11 shows in general, which test tactics can be used depending upon the type of development activity. This is an indicative list and may differ from situation to situation, product to product and customer to customer.

Table 4.11 Development model and testing tactics

Type	Characteristics	Test tactics
Traditional system development such as waterfall model of development	• Uses a system development methodology as defined • User knows requirements completely and these are defined • Development determines structure of application	• Test at end of each task/step/ phase to ensure that exit criteria is achieved • Verify that specifications match user needs exactly • Test functions and structures as per test plan
Iterative development/prototyping development	• Complete set of requirements unknown to users and developers • Structure predefined by development • Refactoring may be done, if required	• Verify that case tools are used properly where required by testing • Test functionality first • Test other areas of product such as integration, system etc
System maintenance methodology	Modify structure if it represents a limiting factor for maintenance activities	• Test structure consistency as it may affect entire application • Works best with release methods of maintenance • Requires heavy regression testing as system is updated
Purchase/contracted software COTS	• System unknown to testers • May contain defects in hidden form • Functionality defined in user manual • Documentation may vary from software to software	• Verify that functionality matches needs of business • Test functionality as per business need • Test fitness for use into production environment

4.12 TESTING PROCESS

Testing is a process made of many milestones. Testers need to achieve them, one by one, to achieve the final goal of testing. Each milestone forms a basis on which the next stage is built. The milestones may vary from organisation to organisation and project to project. Following are few milestones commonly used by many organisations.

Defining Test Policy:

Test policies are defined by senior management of the organisation or test management, and may or may not form a part of the test plan. Test policy at organisation level defines the intent of test management. Test policy is dependent on the maturity of an organisation in development and test process, customer type, type of software, development methodology, etc. Test policy may be tailored at project level, depending upon the scope and historical information available from similar projects.

Defining Test Strategy:

Definition of test strategy includes how the test learn will be organised, and how will it work to achieve test objectives, decision about coverage, automation, etc. Test strategy provide, the actions to the intents defined by test policy. Test strategy helps the test team to understand the approach of testing.

Preparing Test Plan:

Test planning is done by test managers, test leads or senior testers, as the case may be. Test policy sets a tone, whereas test strategy adds the actions required to complete test policy. Test plan tries to answer six basic questions- What, When, Where, Why, Which and How. Test plans are for individual product/project and customer. They are derived from test strategy and give details of execution of testing activity.

Establishing Testing Objectives to Be Achieved:

Test objectives measure the effectiveness and efficiency of a testing process. They also define test achievements that they plan for. Test objectives are also defined from the quality objectives for the project or product, and the critical success factors for testing. Testing objectives must be ‘SMART’ (specific, measurable, agreed upon, realistic, and time bound).

Designing Test Scenarios and Test Cases:

How the test scenarios and test cases will be defined should be explained by the test strategy. A test scenario represents user scenario which acts as a framework for defining test cases. It may have actors and transactions. Similarly, there may be few scenarios arising from system requirements. Theoretically, each transaction in a test scenario eventually becomes a test case.

Writing/Reviewing Test Cases:

Writing and reviewing test cases along with test scenarios and updating requirement traceability matrix accordingly are the tasks done by senior testers or test leads for the project. The traceability matrix gets completed by adding the test cases and finally, the test results. One more column in the traceability matrix would be the results of execution of test cases, i.e., test results.

Defining Test Data:

Test data may be defined on the basis of different techniques available for the purpose. It may include boundary value analysis, error guessing, equivalence partitioning, and state transition. Test data must include valid as well as invalid set of data. It must include some special values generated from error guessing. Test data definitions may be important from testing point of view as different iterations must have different test data sets though it may be used by same test cases.

Creation of Test Bed:

Testing needs creation of environment for testing. It may be a real-life environment or a simulated environment using some simulators. Test bed defines some of the assumptions in a test plan which may induce certain risks of testing. It must reflect real-life situations as closely as possible. Simulators may need definition of few risks, as testing is not done in real environment.

Executing Test Cases:

Execution of actual test cases with the test data defined for testing the software involves applying test cases as well as test data and trying to get the actual results. It sometimes involves updating test cases or test data, if some mistakes are found in initially defined test cases or test data.

Test Result:

Logging results of testing in test log is the last part of the testing iteration. There may be several iterations of testing planned and executed. The defect database may be populated, if expected results are not matching with the actual results. One needs to make sure that the expected results are traceable requirements.

Test Result Analysis:

Testing is a process of SWOT analysis of software under development and testing. Examining test results and analysis may lead to interpretation of software in terms of capabilities and weakness. At the end of testing, the test team must recommend the next step after testing is completed to the project manager- whether the software is ready to go to the next stage or it needs further rework and retesting.

Performing Retesting/Regression Testing When Defects Are Resolved by Development Team:

When defects are given to a development team, they perform analysis and fix the defects. Retesting is done to find out whether the defects declared as fixed and verified by the development team are really fixed or not. Regression testing is done to confirm that the changed part has not affected (in a negative way) any other parts of software, which were working earlier.

Root Cause Analysis and Corrective/Preventive Actions:

Root cause analysis is required to initiate corrective actions. Development/ test team performs post-mortem reviews to understand the weakness of development as well as test process, and initiates actions to improve them. Process improvement is the last activity after the project is closed and formally accepted by the customer. All defect prevention activities must lead to process improvements.

4.13 ATTITUDE TOWARDS TESTING (COMMON PEOPLE ISSUES)

Attitude of development team and senior management or project management towards a test team is a very important aspect to build morale of the test team. It may be initiated from test policy and may be percolated down to test strategy definition and test planning. Some of the views about test team are as follows.

- New members of development team are not accustomed to view testing as a discovery process where defects are found in the product. The defects found are taken as personal blames rather than system/process lacunae. Sometimes, people try to defend themselves, considering it as an attack on the individual.

- 'We take pride on what we developed' or 'we wish to prove that it is right' or 'it is not my fault' are very common responses. Developers may not accept the defect in the first place. If they accept the presence of a defect, then they try to put blame on somebody else. Root cause analysis is very difficult in such situations as people may attach personal ego to it.

- Conflict between developer and tester can create differences between project teams and test teams. In reality, the sole aim of development and testing must be a customer satisfaction, and defects must be considered as something which prevents achievement of this objective.

4.14 TEST METHODOLOGIES/APPROACHES

The two major disciplines in testing are given below:

Black Box Testing:

Black box testing is an attesting methodology where product is tested as per software specifications or requirement statement defined by business analysts/system analysts/customer. Black box testing mainly talks about the requirement specification given by customer, or intended requirements as perceived by testers. It deals with testing of an executable and is independent of platform, database, etc. This testing is with the view as if a user is testing the system.

White Box Testing:

White box testing is a testing methodology where software is tested for the structures, White box testing covers verification of work products as per structure, architecture, coding standards and guidelines of software. It mainly deals with the structure and design of the software product. White box testing requires that testers must have knowledge about development processes and artifacts including various platforms, databases, etc.

There is one more methodology covering both testing methodologies at the same time.

Grey Box Testing:

Grey box testing talks about a combination of both approaches, viz, black box testing and white box testing at the same time. There may be various shades of black box testing as well as white box testing in this type of testing, depending upon the requirements of product. Though not a rule, Grey box testing mainly concentrates on integration testing part along with unit testing.

Broadly, all other testing techniques may be put in any of these three categories, viz. black box testing, white box testing and grey box testing.

4.14.1 Black Box Testing (Domain Testing/Specification Testing):

Black box testing involves testing system/components considering inputs, outputs and general functionalities as defined in requirement specifications. It does not consider any internal processing by the system. Blackbox testing is independent of platform, database, and system to make sure that the system works as per requirements defined as well as implied ones. Actual system (production environment) is simulated, if it is difficult to create a real-life scenario in test laboratory. It does not make any assumption about technicalities of development process, platform, tools, etc. It represents user scenario and actual user interactions.

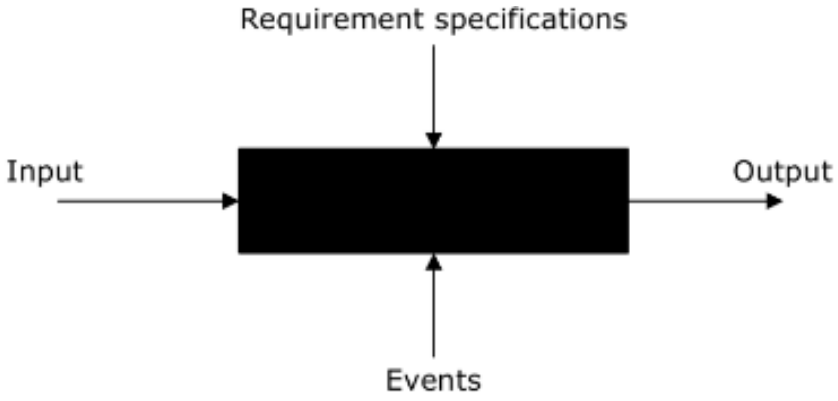


Fig 4.14.1 Black box testing

Fig 4.14.1 shows a block box testing schematically. Black box functional testing is generally conducted for integration testing, system testing, and acceptance testing where the users/customers or the testers as representatives of the customer execute a system as if it is used by users in production environment.

Advantages of 'Black Box Testing':

Black box testing is used primarily to test the behaviour of an application with respect to expressed or implied requirements of the customer.

- Black box testing is the only method to prove that software does what it is supposed to do and it does not do something which can cause a problem to user/customer.
- It is the only method to show that software is living and it really works.
- Some types of testing can be done only by black box testing methodologies, for example, performance and security.

Disadvantages of 'Black Box Testing':

Black box testing has many disadvantages so far as software development methodology is concerned.

- Some logical errors in coding can be missed in black box testing as black box testing efforts are driven by requirements and not by the design. It uses boundary value analysis, equivalence partitioning, and some internal structure problems can be missed.
- Some redundant testing is possible as requirements may execute the same branch of code again and again. If an application calls common functions again and again, then it will be tested so many times that in leads to redundant testing.

Test Case Designing Methodologies:

Black box testing methodology defines how the user is going to interact with the system without any assumption about how the system is built. As

there is no view of how software is built, defining test cases is very difficult. Test cases may be defined using the user scenario called ‘test scenario’. Completeness of test scenario is essential for good testing. Theoretically, each sentence in the test scenario may become a test case. Scenario contains activities in terms of transactions and actors.

Test Data Definition:

Black box testing is mainly driven by the test data used during testing. It may not be feasible to test all possible data which user may be using while working with an application. Some special techniques are applied for defining test data which can give adequate coverage, and also limits the number of test cases and the risk of failure of an application during use. Some of these techniques are mentioned below.

- Equivalence partitioning
- Boundary value analysis
- Cause and effect graph
- State transition testing
- Use case based testing
- Error guessing

4.14.2 White Box Testing:

White box testing is done on the basis of internal structures of software as defined by requirements, designs, coding standards, and guidelines. It starts with reviews of requirements, designs, and codes. White box testing can ensure that relationship between the requirements, designs, and codes can be interpreted. Whitebox testing is mainly a verification technique where one can ensure that software is built correctly. Fig 4.14.2 shows a white box testing schematically.

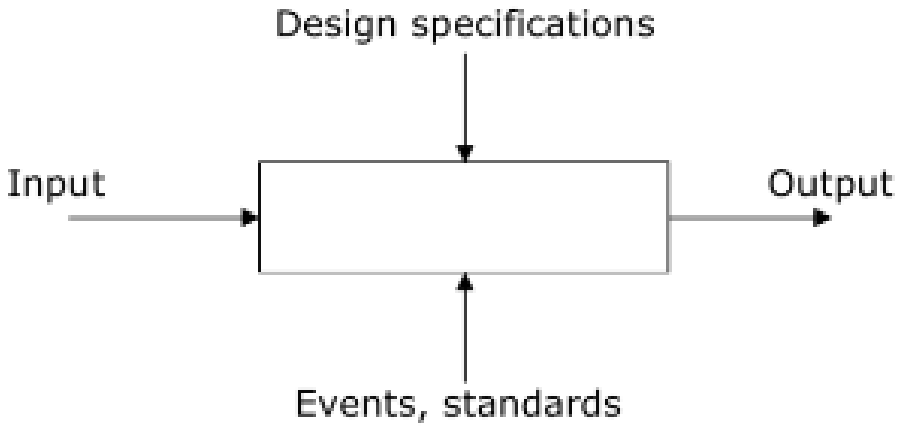


Fig 4.14.2 White box testing

Advantages of 'White Box Testing':

White box testing is a primary method of verification

- Only white box testing can ensure that defined processes, procedures, and methods of development have really been followed during software testing. It can check whether the coding standards, commenting and reuse have been followed or not.
- White box testing of verification can give early warnings, if something is not done properly. It is the most cost-effective way of finding defects as it helps in reducing stage contamination.
- Some characteristics of software work product can be verified only. There is no chance of validating them. For example, code complexity, commenting styles, and reuse.

Disadvantages of 'White Box Testing':

White box testing being a verification technique has few shortcomings.

- It does put ensure that user requirements are met correctly. There is no execution of code, and one does not know whether it will really work or not.
- It does not establish whether decisions, conditions, paths, and statements covered during reviews are sufficient or not for the given set of requirements.
- Sometimes, white box testing is dominated by the usage of checklists. Some defects in checklist may reflect directly in the work product. One must do a thorough analysis of all defects.

Test Case Designing:

Test case designing is based on how test artifacts are created and used during testing. It defines how documents are written and interpreted by each person involved in software development life cycle. Some of the techniques used for white box testing are as follows**.**

- Statement coverage
- Decision coverage
- Condition coverage
- Path coverage
- Logic coverage

4.14.3 Grey Box Testing:

Grey box testing is done on the basis of internal structures of software as defined by requirements, designs, coding standards, and guidelines as well

as the functional and non-functional requirement specifications. Grey box testing combines verification techniques with validation techniques where one can ensure software is built correctly, and also works. Figure 4.14.3 shows a Grey box testing schematically.

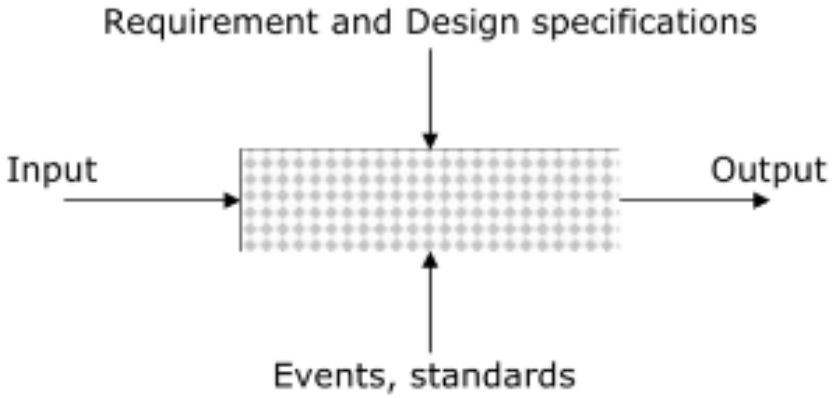


Fig 4.14.3 Grey Box Testing

Advantages of 'Grey Box Testing':

- Grey box testing tries to combine the advantages of white box testing and Black box testing. It checks whether the work product works in a correct manner, both functionally as well as structurally.

Disadvantages of 'Grey Box Testing':

- Generally, Grey box testing is conducted with some automation tools. Knowledge of such tools along with their configuration is essential for performing Grey box testing.

4.15 PEOPLE CHALLENGES IN SOFTWARE TESTING

Testing is a process and must be improved continuously. People need to analyse and take actions on the shortcomings found in the process, so that they can be improved continuously. Few expectations of software process improvement needs from testers are given below.

- The tester is responsible for improving testing process to ensure better products with less number of defects going to customer, thus enhancing customer satisfaction. All defects must be found and the confidence level must be built in the process that can give customer satisfaction. Proper coverage as required by test plan must be achieved.

- Testing needs trained and skilled people who can deliver products with minimum defects to the stakeholders. Testers have to improve their skills through continuous learning.

- The tester needs a positive team attitude for creative destruction of software. Defect in the software is an opportunity to improve the product and not to blame developers. Testers must be able to pinpoint the lacunae in software development process as the defects are found.

- Testing is creative work and a challenging task. Feasible test scenarios and test cases, as well as effective ways of looking for defects are essential to improve testing effectiveness.

- Programmers and testers work together to improve the quality of software developed and delivered to customer, and the process used for software development and testing. The ultimate aim is customer satisfaction.

- Testers hunt for defects they pursue defects not people, including developers. Every defect is considered as a process shortcoming. Defect closure needs retesting and regression testing to find whether the defect is really fixed or not, and to ensure that there is no negative impact of a defect on existing functions.

- Testing needs patience, fairness, ambition, creditability, capability, and diligence on part of testers. Every defect must be seen from the business perspective.

4.16 RAISING MANAGEMENT AWARENESS FOR TESTING

The management must be aware of the roles and responsibilities that testers are performing to achieve customer satisfaction by finding defects. If testers find defects, they can contribute in building good software by reducing probability that customer may find defects.

4.16.1 Tester's Role:

While establishing a test function in an organisation, the management has some objectives to be achieved. Test team needs to understand these objectives and fulfil them.

- Calculate testing cost, effectiveness of testing and ensure that management understands the same. By doing good testing, number of customer complaints must reduce and cost of failure must go down.

- Demonstrate cost reduction and increase in effectiveness over a time span (as rework and scrap reduce over a long horizon). This can be shown by reduced customer complaints as well as less rework.

- Highlight needs and benefits of training-in test team as well as development team-on testing activities and skills, so that testers can perform better. Many developers need information about unit testing, integration testing and their role in such testing.

- Collect and distribute information on testing to all team members as well as development team/organization which can be used for improvement.

- Get involved in test budgeting. Testing needs people, money, time, training and other resources. The organization may have to develop budget to procure all these aspects.

4.17 SKILLS REQUIRED BY TESTER

Testing needs a disciplined approach. A tester is the person entrusted by an organisation to work as the devil's agent. He/she is a person working for the client, finding the obvious defects in the processes and products. The main purpose of testing is to demonstrate that defects are present, and point towards the weaker areas in the software as well as processes used to build it, so that actions can be initiated in that direction. It must build confidence in management and customer that the application with which they will be working is usable and does not have defects. One must try to build maximum possible skills, and training is one of the effective methods to build a good testing team.

4.17.1 General Skills:

Written and Verbal Presentation Skill:

Presenting test results or discussing about an application or defects involves communication with many people. Testers are supposed to present test results and tell development team, customer and management about the present status of application and where further improvements can be done. Testers must be good in presentation skills.

Effective Listening Skill:

Testers need to listen to the customers' voice as well as views of developers. Listening to customer as well as developer- to understand the needs and requirements correctly is required to ensure that the scenarios and test cases can be written in a proper way. Tester's listening skills can convert testing into effective testing. Listening skills can give them complete information about the process, software application and also what the customer and management are looking for.

Facilitation Skill:

Facilitation of development team as well as customer is done by testers, so that defects are taken in the proper spirit. Testers must be able to tell the exact nature of a defect, how it is happening and how it will affect the users. Testers must contribute to improve development process and take part in building better product.

Software Development, Operations and Maintenance:

Good knowledge of software development life cycle and software testing life cycle help testers in designing test scenarios and test cases accordingly. The defect age and cost of testing are important parameters to be controlled by testers.

Continuous Education:

Testers must undergo continuous education and training to build and enforce quality practices in development processes. They need to undergo training for test planning, test case definition, test data definition, methods and processes applied for testing, and reporting defects.

4.17.2 Testing Skills:

Concepts of Testing:

A tester must have complete knowledge about testing as a discipline. He/she must understand methods, processes, and concepts of testing. He/she must be capable of doing test planning, designing test scenario, writing test cases, and defining test strategy, and defining test data.

Levels of Testing:

Testing is a multitier activity where the application goes from one level to another after successful completion of the previous level. Testers are involved in each phase of software development right from proposal and contract, followed by requirement till acceptance testing. Testers must ensure that each phase is passed successfully.

Techniques for Validation and Verification:

Techniques of verification/validation must be understood and facilitated by testers to the development team, customer and management. While writing test cases, he/she needs to define test case pass/fail criteria to validate the test case and product.

Selection and Use of Testing Tools:

Testing involves use of various tools including automation tools, defect tracking tools, configuration management tools, and simulators. A tester must understand and use the tools effectively.

Knowledge of Testing Standards:

Testing standards are defined by software quality management. There can be some international standards or organisation/customer defined standards. Testers need to understand these standards, and apply them effectively so that a common understanding can be achieved.

Risk Assessment and Management:

Testing is risk-driven activity. Test cases and test data must be defined to minimise risks to the final users in production environment. Testing efforts must be managed to improve their effectiveness and efficiency. Managing testing involves planning, organising, directing, coordinating, and controlling testing process.

Developing Test Plan:

Test plan development is generally done by test managers or test leads while implementation of these plans is done by testers. Individual testers must plan for their part in overall test plan for the project.

Defining Acceptance Criteria:

Definition of acceptance criteria is an important milestone for testing. Generally, acceptance criteria are defined by customer well before the project starts. Testers need to define acceptance criteria for the phases and iterations of testing. There are various forms of acceptance criteria which will be discussed later. The phase-end acceptance criteria may be defined by the testers in test plan.

Checking of Testing Processes:

Testers follow the processes as defined in the test plan. They need to audit the testing processes to check the compliance and effectiveness, and also initiate actions if deviations are observed. Testers must contribute in testing process improvements.

Execution of Test Plan:

Testers are given the responsibility of executing a test plan. It includes defining test scenario, test cases, and test data, and their execution. They put test results in test log and defects in defect logging tool. Resolved defects are taken for retesting. They must perform regression testing when planned. Testers must do analysis of test results to define weaker and stronger areas of software development and testing process. They must be able to define test coverage such as code coverage, statement cover, branch coverage, requirement coverage, and function coverage.

Continuous Improvement of Testing Process:

Testers must plan for continuous improvement of testing process. Testing process must be subjected to improvements followed by phase of consolidations. Actions must be planned for improving process, and the results must be compared with expectations. If some deviations are observed, new actions can be initiated.

4.18 SOFTWARE LIFE CYCLE

SDLC is a process that defines the various stages involved in the development of software for delivering a high-quality product. SDLC stages cover the complete life cycle of a software i.e. from inception to retirement of the product.

Adhering to the SDLC process leads to the development of the software in a systematic and disciplined manner.

Purpose:

Purpose of SDLC is to deliver a high-quality product which is as per the customer's requirement.

SDLC has defined its phases as, Requirement gathering, Designing, Coding, Testing, and Maintenance. It is important to adhere to the phases to provide the Product in a systematic manner.

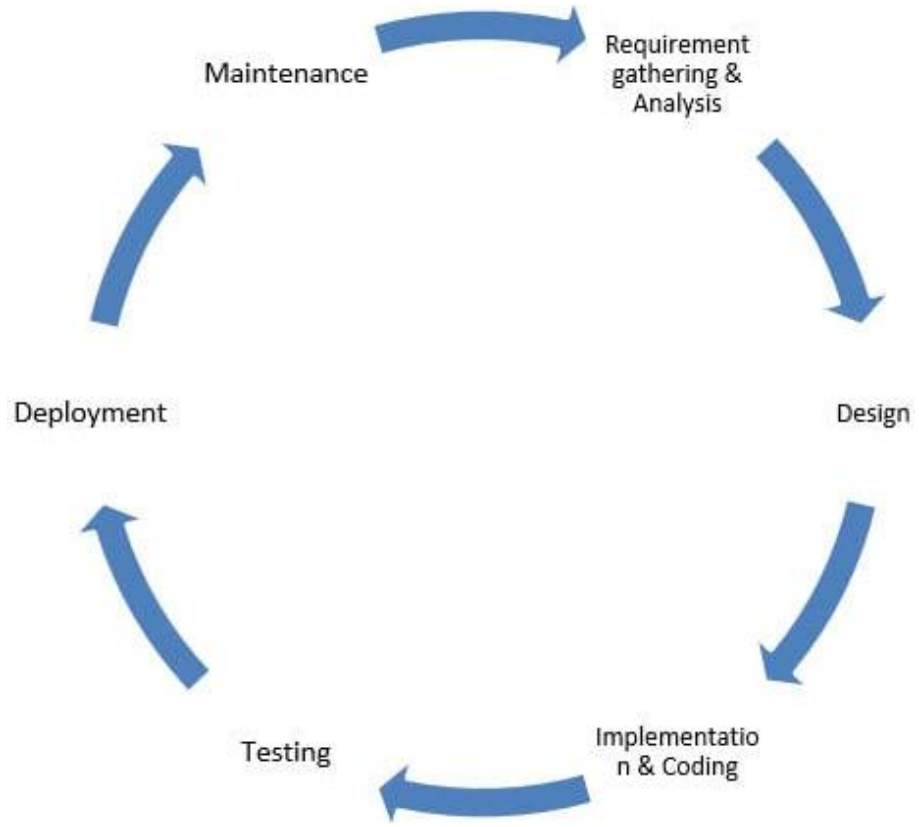
For Example, A software has to be developed and a team is divided to work on a feature of the product and is allowed to work as they want. One of the developers decides to design first whereas the other decides to code first and the other on the documentation part.

This will lead to project failure because of which it is necessary to have a good knowledge and understanding among the team members to deliver an expected product.

SDLC Cycle:

SDLC Cycle represents the process of developing software.

Below is the diagrammatic representation of the SDLC cycle:



SDLC Phases

Given below are the various phases:

- Requirement gathering and analysis

- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

1) Requirement Gathering and Analysis:

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

For Example, A customer wants to have an application which involves money transactions. In this case, the requirement has to be clear like what kind of transactions will be done, how it will be done, in which currency it will be done, etc.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

2) Design:

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

3) Implementation or Coding:

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

4) Testing:

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer SRS document to make sure that the software is as per the customer's standard.

5) Deployment:

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

6) Maintenance:

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

4.19 SOFTWARE DEVELOPMENT MODELS

A software life cycle model is a descriptive representation of the software development cycle. SDLC models might have a different approach but the basic phases and activity remain the same for all the models.

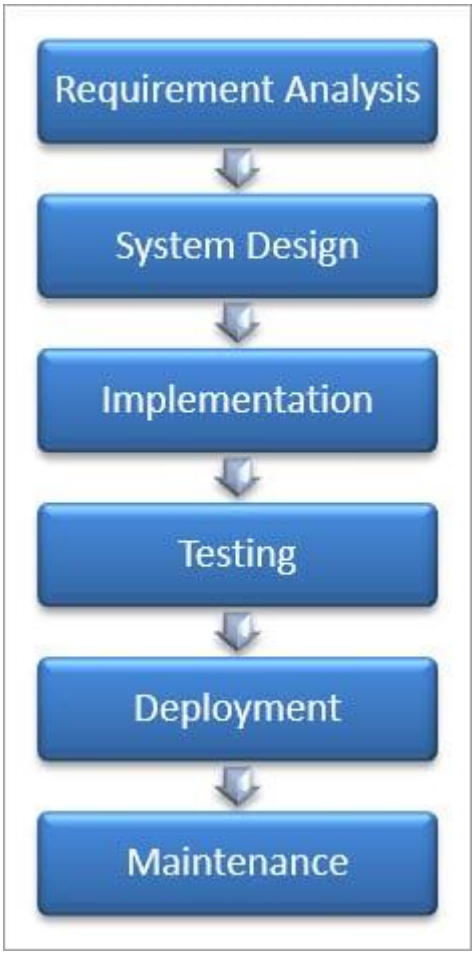
1) Waterfall Model:

Waterfall model is the very first model that is used in SDLC. It is also known as the linear sequential model.

In this model, the outcome of one phase is the input for the next phase. Development of the next phase starts only when the previous phase is complete.

- First, Requirement gathering and analysis is done. Once the requirement is freeze then only the System Design can start. Herein, the SRS document created is the output for the Requirement phase and it acts as an input for the System Design.
- In System Design Software architecture and Design, documents which act as an input for the next phase are created i.e. Implementation and coding.
- In the Implementation phase, coding is done and the software developed is the input for the next phase i.e. testing.
- In the testing phase, the developed code is tested thoroughly to detect the defects in the software. Defects are logged into the defect tracking tool and are retested once fixed. Bug logging, Retest, Regression testing goes on until the time the software is in go-live state.

- In the Deployment phase, the developed code is moved into production after the sign off is given by the customer.
- Any issues in the production environment are resolved by the developers which come under maintenance.



Advantages of the Waterfall Model:

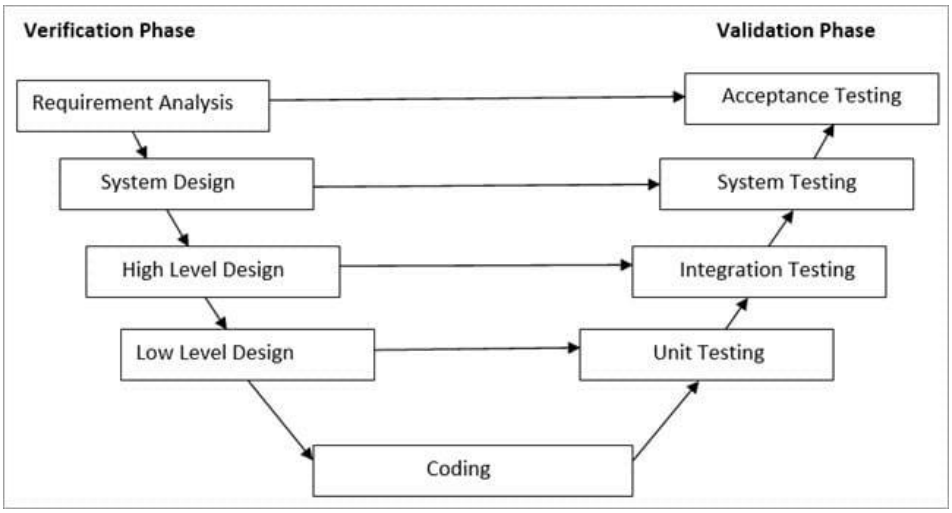
- Waterfall model is the simple model which can be easily understood and is the one in which all the phases are done step by step.
- Deliverables of each phase are well defined, and this leads to no complexity and makes the project easily manageable.

Disadvantages of Waterfall model:

- Waterfall model is time-consuming & cannot be used in the short duration projects as in this model a new phase cannot be started until the ongoing phase is completed.
- Waterfall model cannot be used for the projects which have uncertain requirement or wherein the requirement keeps on changing as this model expects the requirement to be clear in the requirement gathering and analysis phase itself and any change in the later stages would lead to cost higher as the changes would be required in all the phases.

2) V-Shaped Model:

V- Model is also known as Verification and Validation Model. In this model Verification & Validation goes hand in hand i.e. development and testing goes parallel. V model and waterfall model are the same except that the test planning and testing start at an early stage in V-Model.



a) Verification Phase:

(i) Requirement Analysis:

In this phase, all the required information is gathered & analyzed. Verification activities include reviewing the requirements.

(ii) System Design:

Once the requirement is clear, a system is designed i.e. architecture, components of the product are created and documented in a design document.

(iii) High-Level Design:

High-level design defines the architecture/design of modules. It defines the functionality between the two modules.

(iv) Low-Level Design:

Low-level Design defines the architecture/design of individual components.

(v) Coding:

Code development is done in this phase.

b) Validation Phase:

(i) Unit Testing:

Unit testing is performed using the unit test cases that are designed and is done in the Low-level design phase. Unit testing is performed by the

developer itself. It is performed on individual components which lead to early defect detection.

(ii) Integration Testing:

Integration testing is performed using integration test cases in High-level Design phase. Integration testing is the testing that is done on integrated modules. It is performed by testers.

(iii) System Testing:

System testing is performed in the System Design phase. In this phase, the complete system is tested i.e. the entire system functionality is tested.

(iv) Acceptance Testing:

Acceptance testing is associated with the Requirement Analysis phase and is done in the customer's environment.

Advantages of V – Model:

- It is a simple and easily understandable model.
- V –model approach is good for smaller projects wherein the requirement is defined and it freezes in the early stage.
- It is a systematic and disciplined model which results in a high-quality product.

Disadvantages of V-Model:

- V-shaped model is not good for ongoing projects.
- Requirement change at the later stage would cost too high.

4.20 TEST LEVELS

There are mainly four testing levels are:

i) Unit Testing

ii) Integration Testing

iii) System Testing

iv) Acceptance Testing

Unit Testing:

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing:

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that a developer can use to verify a source code. After having exhausted all the options, there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing:

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

S. No.	Integration Testing Method
1	**Bottom-up integration:** This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
2	**Top-down integration:** In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic actual situations.

System Testing:

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

System testing is important because of the following reasons:

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.

- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Regression Testing:

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Regression testing is important because of the following reasons:

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Test coverage is increased without compromising timelines.
- Increase speed to market the product.

Acceptance Testing:

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.

By performing acceptance tests on an application, the testing team will reduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

Alpha Testing:

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and

system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application −

- Spelling Mistakes
- Broken Links
- Cloudy Directions
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing:

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as **pre-release testing**. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following −

- Users will install, run the application and send their feedback to the project team.
- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release it to the general public will increase customer satisfaction.

4.21 TEST TYPES

A test type is a group of test activities aimed at testing specific characteristics of a software system, or a part of a system, based on specific test objectives. Such objectives may include:

Evaluating functional quality characteristics, such as completeness, correctness, and appropriateness

Evaluating non-functional quality characteristics, such as reliability, performance efficiency, security, compatibility, and usability

Evaluating whether the structure or architecture of the component or system is correct, complete, and as specified

Evaluating the effects of changes, such as confirming that defects have been fixed (confirmation testing) and looking for unintended changes in behavior resulting from software or environment changes (regression testing)

i) Functional Testing:

Functional testing of a system involves tests that evaluate functions that the system should perform. Functional requirements may be described in work products such as business requirements specifications, epics, user stories, use cases, or functional specifications, or they may be undocumented. The functions are "what" the system should do.

Functional tests should be performed at all test levels. Functional testing considers the behavior of the software

ii) Non-functional Testing:

Non-functional testing of a system evaluates characteristics of systems and software such as usability, performance efficiency or security. Non-functional testing is the testing of "how well" the system behaves. Non-functional testing can and often should be performed at all test levels, and done as early as possible. The late discovery of non-functional defects can be extremely dangerous to the success of a project.

iii) Structural Testing:

Structural Testing derives tests based on the system's internal structure or implementation. Internal structure may include code, architecture, work flows, and/or data flows within the system.

iv) Change-related Testing:

When changes are made to a system, either to correct a defect or because of new or changing functionality, testing should be done to confirm that the changes have corrected the defect or implemented the functionality correctly, and have not caused any unforeseen adverse consequences.

Confirmation Testing:

After a defect is fixed, the software may be tested with all test cases that failed due to the defect, which should be re-executed on the new software version.

Regression Testing:

It is possible that a change made in one part of the code, whether a fix or another type of change, may accidentally affect the behavior of other parts of the code, whether within the same component, in other components of the same system, or even in other systems.

Changes may include changes to the environment, such as a new version of an operating system or database management system. Such unintended side-effects are called regressions.

Regression testing involves running tests to detect such unintended side effects. Confirmation testing and regression testing are performed at all test levels.

Regression test suites are run many times and generally evolve slowly, so regression testing is a strong candidate for automation. Automation of these tests should start early in the project.

4.22 TARGETS OF TESTING

The main goal of software testing is to find bugs as early as possible and fix bugs and make sure that the software is bug-free. The goals of software testing may be classified into three major categories as follows:

1. **Immediate Goals**
2. **Long-term Goals**
3. **Post-Implementation Goals**

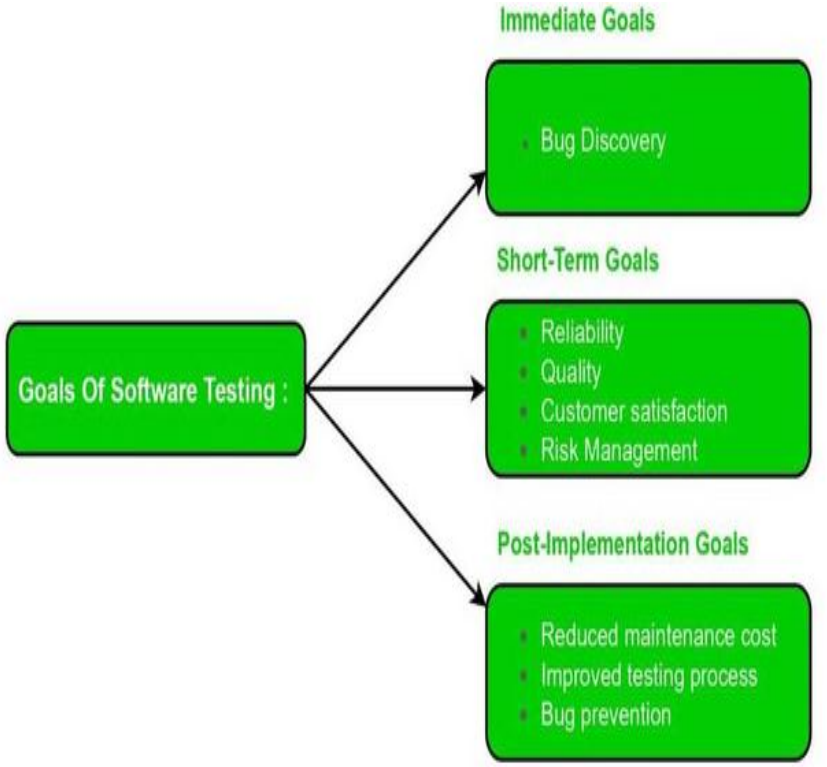


Fig : Software Testing Goals

1. Immediate Goals:

These objectives are the direct outcomes of testing. These objectives may be set at any time during the SDLC process. Some of these are covered in detail below:

- **Bug Discovery:** This is the immediate goal of software testing to find errors at any stage of software development. The number of

bugs is discovered in the early stage of testing. The primary purpose of software testing is to detect flaws at any step of the development process. The higher the number of issues detected at an early stage, the higher the software testing success rate.

- **Bug Prevention:** This is the immediate action of bug discovery, that occurs as a result of bug discovery. Everyone in the software development team learns how to code from the behavior and analysis of issues detected, ensuring that bugs are not duplicated in subsequent phases or future projects.

2. Long-Term Goals:

These objectives have an impact on product quality in the long run after one cycle of the SDLC is completed. Some of these are covered in detail below:

- **Quality:** This goal enhances the quality of the software product. Because software is also a product, the user's priority is its quality. Superior quality is ensured by thorough testing. Correctness, integrity, efficiency, and reliability are all aspects that influence quality. To attain quality, you must achieve all of the above-mentioned quality characteristics.

- **Customer Satisfaction:** This goal verifies the customer's satisfaction with a developed software product. The primary purpose of software testing, from the user's standpoint, is customer satisfaction. Testing should be extensive and thorough if we want the client and customer to be happy with the software product.

- **Reliability:** It is a matter of confidence that the software will not fail. In short, reliability means gaining the confidence of the customers by providing them with a quality product.

- **Risk Management:** Risk is the probability of occurrence of uncertain events in the organization and the potential loss that could result in negative consequences. Risk management must be done to reduce the failure of the product and to manage risk in different situations.

3. Post Implemented Goals:

After the product is released, these objectives become critical. Some of these are covered in detail below:

- **Reduce Maintenance Cost:** Post-released errors are costlier to fix and difficult to identify. Because effective software does not wear out, the maintenance cost of any software product is not the same as the physical cost. The failure of a software product due to faults is the only expense of maintenance. Because they are difficult to discover, post-release mistakes always cost more to rectify. As a result, if testing is done thoroughly and effectively, the risk of failure is lowered, and maintenance costs are reduced as a result.

- **Improved Software Testing Process:** These goals improve the testing process for future use or software projects. These goals are known as post-implementation goals. A project's testing procedure may not be completely successful, and there may be room for improvement. As a result, the bug history and post-implementation results can be evaluated to identify stumbling blocks in the current testing process that can be avoided in future projects.

4.23 MAINTENANCE TESTING

Maintenance Testing is also known as post-release software testing. This is a type of software testing that takes place when the software has been released into production and any changes have been made to fix bugs or add new features to the existing system.

Maintenance Testing provides feedback on how well the latest release is working in real life, whether it solves the problems identified by pre-release testing. This also helps developers find any new bugs that may have been introduced by the corrective and emergency changes in development, and it ensures that these don't break functionality with other parts of a system. This testing is performed at no cost to clients who contract for post-release updates or maintenance support from software vendors.

This can be performed at any time after release: it doesn't have to wait until another major version update or has been released.

Maintenance Testing is performed by the same team of testers who perform Pre-release testing to already deployed software, and it usually requires a similar level of skills from the tester as well.

This can also be used to test new releases of a software product that has not yet been released to the existing system before any major changes are made and without the need for an expensive re-testing phase.

Maintenance Testing often takes place on local servers or in an operational environment in an effort to mimic production environments more accurately and for long periods of time with high usage rates. Maintenance Testers also often have to perform regression testing, which means they may need to retest features and existing operational system parts of the software that were already tested before a new release.

It is usually provided for free by vendors who offer post-release updates or maintenance support contracts in order for their customers to identify bugs quickly when system changes are made in the existing software. Maintenance Testing is also often used by SMEs who want to release their products more quickly than would be possible with traditional testing.

Why Maintenance Testing is required in Software Testing?

Maintenance Testing is required for a few reasons.

- It's a good idea to do Maintenance Testing before major changes are made to existing software and without the need for an expensive re-testing phase.
- It ensures that changes made during post-release development are not causing problems with other parts of the system, which can be hard to detect before release without enough hours spent in regression testing.
- In regression testing, testers retest features and parts of the software that were already tested before a new release.
- Bug fixes won't work properly if they conflict with new features introduced by the new release.
- This is done by the same team of testers who performed Pre-release testing, and it usually requires a similar level of skills from the tester as well for these reasons.
- It is usually provided for free by vendors who offer post-release updates or maintenance support contracts in order for their customers to identify bugs quickly when system changes are made.

I hope you found this post about Why Should You Do Maintenance Testing interesting! If so, please sign up for my newsletter to receive notifications of future posts.

Types of Maintenance Testing:

When the maintenance testers are validating the application, they need to consider two things. Based on the test types

- **Confirmation Testing:** On this confirmation maintenance testing, the Testers or QA have to mainly focus on the modified functionalities. They have to verify every aspect of the application is working as it should be.
- **Regression Testing:** Testing the existing functionality to ensure that it is not broken or degraded by the new functionality.

Benefits of maintenance testing:

By doing regression maintenance testing it helps catch bugs before the pre-release stage, ensuring major changes in development don't break other parts of the system without regression maintenance testing, and it ensures bug fixes work properly with new features.

Risks of not doing Maintenance testing:

When not testing changes before release they can break other parts of the system, bug fixes won't work properly with new features introduced in new releases, and will require more time to fix them if regression maintenance testing was done.

in simple words, If it is not done, bugs will be missed and the fixes won't work with new features.

4.24 TESTING TIPS

Customer pays for a product on the basis of the value he finds in acquiring such product. Cost of development and cost of testing define the profit available to an organization by selling such product project. Market forces define the sales price of the product project. There is always a pressure on development and testing to reduce the cost to improve profitability. The following list may be considered as a generic guideline for reducing com of testing or improving the value of a product.

Reduce Software Development Risk:

Development activities may introduce several risks starting from requirement capturing, through design and development, coding and so on. Software testing must be effective to locate the defects as early as possible so that stage contamination can be reduced.

Perform Testing Effectively:

Testing must be able to capture defects as effectively and efficiently as possible. It must give adequate confidence to users that application will not meet any accidental failures. It must be able to find as many defects as possible, so that they will be fixed eventually and probability of failure at customer place is minimised.

Uncover Maximum Number of Defects:

Each defect uncovered must reduce a chance of customer complaint. If testing can find 100% defects present in the given software, it will not be possible for customer to see any failure during use. Though it is very difficult, one must try to find all conceivable defects. Successful tester is one who finds maximum number of defects.

Use Business Logic:

Testers must have a good knowledge about the domain under testing. This is true for system testers where it is expected that they would be working as normal users. They must use business logic to improve efficiency and effectiveness of testing, and also testing must give enough confidence to users that there will not be any accidental failures.

Testing Must Occur Throughout SDLC:

Defect found as early as possible can reduce cost of failure by preventing stage contamination. Testing concentrating more on system testing may not be very effective as software developed may be very fragile.

Testing Must Cover Functional/Structural Parts:

Often, people consider functional testing as a complete testing. One must keep in mind that there are five types of requirements mentioned by 'TELOS' (Technical Economic Legal Operational System). Only operational requirements may cover functional as well as non-functional requirements. It must also cover the way software is built to give better screen designs, optimum performance, and security.

4.25 SUMMARY

This chapter presents the definitions of a successful tester and the basic principles of software testing. It offers a detailed exposition of the process of creating test policy, test strategy, and test plan. It also gives an in depth understanding of 'Black Box Testing', 'White Box Testing' and 'Gray Box Testing'. The chapter concludes with skills required by a good tester and challenges faced by a tester.

4.26 EXERCISES

1) Explain the concept of test team's defect finding efficiency.
2) Explain test case's defect finding efficiency.
3) What are the challenges faced by testers?
4) Explain the process of developing test strategy.
5) Explain the process of developing test methodology.
6) Which skills are expected in a good tester?

4.27 REFERENCES

- Software Testing and Continuous Quality Improvement by William E. Lewis CRC Press Third 2016.
- Software Testing: Principles, Techniques and Tools M. G. Limaye TMH 2017.
- Foundations of Software Testing Dorothy Graham, Erik van Veenendaal, Isabel Evans, and Rex Black Cengage Learning 3 rd.
- Software Testing: A Craftsman's Approach Paul C. Jorgenson CRC Press 4th 2017.
- https://www.softwaretestinghelp.com/writing-test-strategy-document-template/
- https://www.softwaretestinghelp.com/10-qualities-that-can-make-you-a-good-tester/

*****

UNIT - III

5

UNIT TESTING: BOUNDARY VALUE TESTING, EQUIVALENCE CLASS TESTING

Unit Structure

5.0 OBJECTIVES

This chapter will make the readers understand the following concepts:

- How software works
- Test cases
- Dynamic and Static Testing
- Boundary value Analysis
- Equivalence class portioning
- Examples of Boundary value Analysis
- and Equivalence class portioning

5.1 HOW DOES SOFTWARE TESTING WORK?

Software testing is the process of evaluating and verifying that a software product or application does what it is supposed to do. The benefits of testing include preventing bugs, reducing development costs and improving performance. The process or method of finding error/s in a software application or program so that the application functions according to the end user's requirement is called software testing.

Description:

Software testing is the process of verifying a system with the purpose of identifying any errors, gaps or missing requirement versus the actual requirement. Software testing is broadly categorised into two types - functional testing and non-functional testing.

When to start test activities: Testing should be started as early as possible to reduce the cost and time to rework and produce software that is bug-free so that it can be delivered to the client. However, in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the software is out there in productions. It also depends on the development model that is being used. For example, in the Waterfall model, testing starts from the testing phase which is quite below in the tree,; but in the V-model, testing is performed parallel to the development phase.

5.1.1 Importance of Software Testing:

Few can argue against the need for quality control when developing software. Late delivery or software defects can damage a brand’s reputation - leading to frustrated and lost customers. In extreme cases, a bug or defect can degrade interconnected systems or cause serious malfunctions.

Consider Nissan having to recall over 1 million cars due to a software defect in the airbag sensor detectors. Or a software bug that caused the failure of a USD 1.2 billion military satellite launch. 2 The numbers speak for themselves. Software failures in the US cost the economy USD 1.1 trillion in assets in 2016. What's more, they impacted 4.4 billion customers. 3

Though testing itself costs money, companies can save millions per year in development and support if they have a good testing technique and QA processes in place. Early software testing uncovers problems before a product goes to market. The sooner development teams receive test feedback, the sooner they can address issues such as:

- Architectural flaws
- Poor design decisions
- Invalid or incorrect functionality
- Security vulnerabilities
- Scalability issues

When development leaves ample room for testing, it improves software reliability and high-quality applications are delivered with few errors. A system that meets or even exceeds customer expectations leads to potentially more sales and greater market share.

5.1.2 What Are The Categories of Test Design Techniques?

A test design technique basically helps us to select a good set of tests from the total number of all possible tests for a given system. There are many different types of software testing technique, each with its own strengths and weaknesses. Each individual technique is good at finding particular types of defect and relatively poor at finding other types.

For example, a technique that explores the upper and lower limits of a single input range is more likely to find boundary value defects than defects associated with combinations of inputs. Similarly, testing performed at different stages in the software development life cycle will find different types of defects; component testing is more likely to find coding logic defects than system design defects.

Each testing technique falls into one of a number of different categories. Broadly speaking there are two main categories:

1. Static technique
2. Dynamic technique

- Specification-based (black-box, also known as behavioral techniques)

- Structure-based (white-box or structural techniques)
- Experience- based

Dynamic techniques are subdivided into three more categories: specification-based (black-box, also known as behavioral techniques), structure-based (white-box or structural techniques) and experience-based. Specification-based techniques include both functional and nonfunctional techniques (i.e. quality characteristics).

Dynamic Testing is a software testing method used to test the dynamic behaviour of software code. The main purpose of dynamic testing is to test software behaviour with dynamic variables or variables which are not constant and finding weak areas in software runtime environment. The code must be executed in order to test the dynamic behaviour.

We all know that Testing is verification and validation, and it takes 2 Vs to make testing complete. Out of the 2 Vs, Verification is called a Static testing and the other "V", Validation is known as Dynamic testing.

Let's understand How to do Dynamic Testing with an example:

Suppose we are testing a Login Page where we have two fields say "Username" and "Password" and the Username is restricted to Alphanumeric.

When the user enters Username as "Guru99", the system accepts the same. whereas when the user enters as Guru99@123 then the application throws an error message. This result shows that the code is acting dynamically **based on the user input.**

Dynamic testing is when you are working with the actual system by providing an input and comparing the actual behaviour of the application to the expected behaviour. In other words, working with the system with the intent of finding errors.

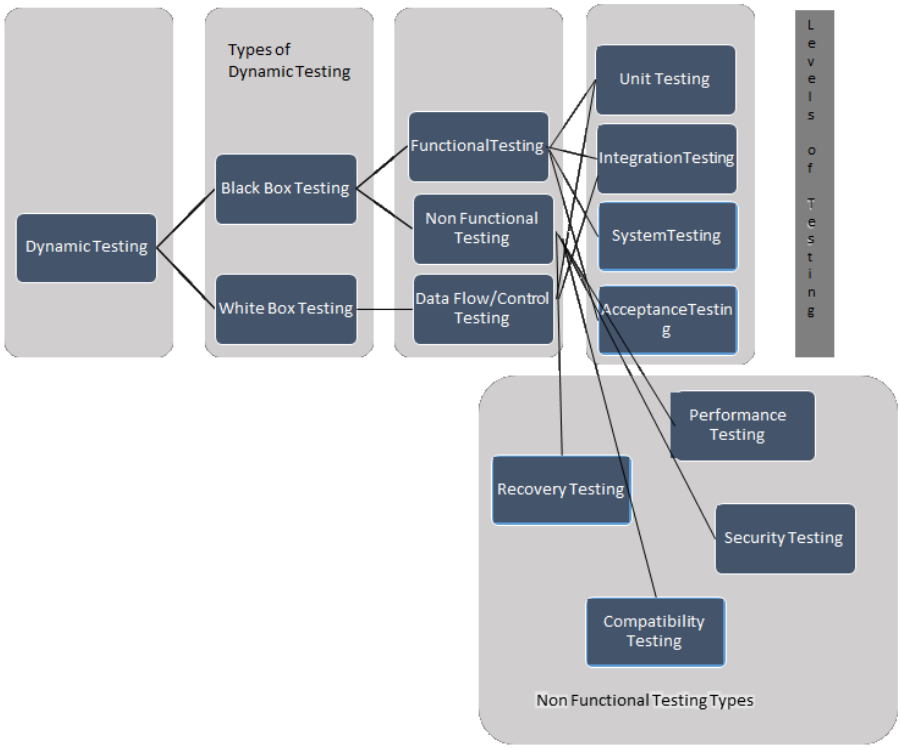
So based on the above statements we can say or conclude that dynamic testing is a process of validating software applications as an end user under different environments to build the right software.

5.1.3 Types of Dynamic Testing:

Dynamic Testing is classified into two categories:

- White Box Testing
- Black Box Testing

The below pictorial representation gives us an idea about types of Dynamic Testing, Levels of Testing, etc.



What is Static testing technique?

- Static testing is the testing of the software work products manually, or with a set of tools, but they are **not executed**.
- It starts early in the Life cycle and so it is done during the verification process.
- It does not need computer as the testing of program is done without executing the program. For example: reviewing, walk through, inspection, etc.
- Most static testing techniques can be used to 'test' any form of document including source code, design documents and models, functional specifications and requirement specifications.

What is Dynamic testing technique?

- This testing technique needs computer for testing.
- It is done during Validation process.
- The software is tested by executing it on computer. Ex: Unit testing, integration testing, and system testing.

5.2 BLACK BOX TESTING

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

5.2.1 Generic Steps of Black Box Testing:

- The black box test is based on the specification of requirements, so it is examined in the beginning.
- In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- The fourth phase includes the execution of all test cases.
- In the fifth step, the tester compares the expected output against the actual output.
- In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

5.2.2 Test Procedure:

The test procedure of black box testing is a kind of process in which the tester has specific knowledge about the software's work, and it develops test cases to check the accuracy of the software's functionality.

It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function. A tester knows about the definite output of a particular input, but not about how the result is arising. There are various techniques used in black box testing for testing like decision table technique, boundary value analysis technique, state transition, All-pair testing, cause-effect graph technique, equivalence partitioning technique, error guessing technique, use case technique and user story technique.

Test cases:

Test cases are created considering the specification of the requirements. These test cases are generally created from working descriptions of the software including requirements, design parameters, and other specifications. For the testing, the test designer selects both positive test scenario by taking valid input values and adverse test scenario by taking invalid input values to determine the correct output. Test cases are mainly designed for functional testing but can also be used for non-functional testing. Test cases are designed by the testing team; there is not any involvement of the development team of software.

5.2.3 Types Of Black Box Testing:

There are many types of Black Box Testing but the following are the prominent ones:

Functional testing:

This black box testing type is related to the functional requirements of a system; it is done by software testers.

Non-functional testing:

This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.

Regression testing:

Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Tools used for Black Box Testing:

Tools used for Black box testing largely depend on the type of black box testing you are doing.

- For Functional/ Regression Tests you can use – QTP, Selenium
- For Non-Functional Tests, you can use – LoadRunner, Jmeter

5.2.4 TECHNIQUES USED IN BLACK BOX TESTING

Decision Table Technique	Decision Table Technique is a systematic approach where various input combinations and their respective system behavior are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs.
Boundary Value Technique	Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering

	boundary value whether the software is producing correct output or not.
State Transition Technique	State Transition Technique is used to capture the behavior of the software application when different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to access the application.
All-pair Testing Technique	All-pair testing Technique is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc.
Cause-Effect Technique	Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result. It is based on a collection of requirements.
Equivalence Partitioning Technique	Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.
Error Guessing Technique	Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software.
Use Case Technique	Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end.

Practically, due to time and budget considerations, it is not possible to perform exhausting testing for each set of test data, especially when there is a large pool of input combinations.

- We need an easy way or special techniques that can select test cases intelligently from the pool of test-case, such that all test scenarios are covered.
- We use techniques such as – **Equivalence Partitioning & Boundary Value Analysis testing, Decision table-based testing, path testing, Data flow testing techniques** to achieve this.

5.3 BOUNDARY VALUE TESTING

Boundary value analysis is a software testing technique in which tests are designed to include representatives of boundary values. The idea comes from the Boundary (topology).

A greater number of errors occur at the boundaries of the input domain rather than in the "center"

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

It selects test cases at the edges of a class

It derives test cases from both the input domain and output domain

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called “boundary testing”.

Advantagesof BoundaryValueTesting:

The BVA technique of testing is quite easy to use and remember because of the uniformity of identified tests and the automated nature of this technique.

One can easily control the expenses made on the testing by controlling the number of identified test cases. This can be done with respect to the demand ofthe software that needs to be tested.

BVA is the best approach in cases where the functionality of a software is basedon numerous variables representing physical quantities.

The technique is best at revealing any potential UI or user input troubles in thesoftware.

The procedure and guidelines are crystal clear and easy when it comes to determining the test cases through BVA.

The test cases generated through BVA are very small.

DisadvantagesofBoundaryValueTesting:

This technique sometimes fails to test all the potential input values.And so, the results are unsure.

The dependencies with BVA are not tested between two inputs.

This technique doesn’t fit well when it comes to Boolean Variables.

It only works well with independent variables that depict quantity.

Types of Boundary Value Testing:

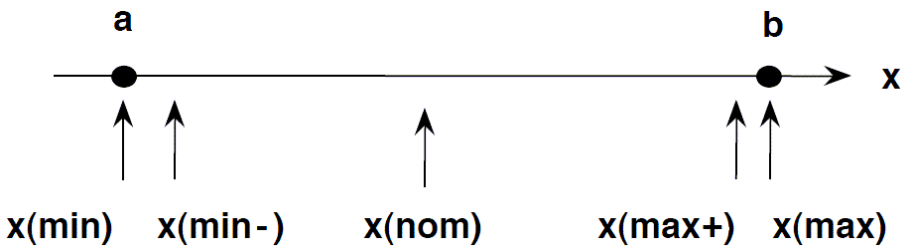
- Normal Boundary Value Testing
- Robust Boundary Value Testing
- Worst-case Boundary Value Testing
- Robust Worst-case Boundary Value Testing

5.4 NORMAL BOUNDARY VALUE ANALYSIS

Normal Boundary value analysis is a black-box testing technique, closely associated with equivalence class partitioning. In this technique, we analyze the behavior of the application with test data residing at the boundary values of the equivalence classes.

The basic idea in normal boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum



5.5 ROBUST BOUNDARY VALUE TESTING

In robustness testing, the software is tested by giving invalid values as inputs. Robustness testing is usually done to test exception handling.

In robust boundary value testing, we make combinations in such a way that some of the invalid values are also tested as input.

The type of testing done by causing the software or system to fail in order to test the robustness is called robustness testing.

Robust Boundary value testing on 3 variables:

Suppose we have 3 variables X, Y and Z to test

The range of X: 0 to 100

The range of Y: 20 to 60

The range of Z: 80 to 100

Testing points detected in Simple Robust Boundary Value Testing

	X	y	z
Min-	-1	19	79
Min	0	20	80
Min+	1	21	81
Nominal	50	40	90
Max-	99	59	99
Max	100	60	100
Max+	101	61	101

Test Cases:

Total Test cases =(Number of variables * Number of testing points without nominal)+ (1 for Nominal)

These testing points are min-, min, min+, max- and max and max+19=(3*6)+1

We can generate 19 test cases from both variables X, Y, and Z.

- There are a total of 3 variables X, Y and Z
- There are 6 possible values like min-, min, min+, max-, max and max+
- 1 is for nominal

Logic:

When we make test cases, we will fix the nominal value of the two variables and change the values of the third variable.

For example

- We will fix the nominal values of X and Y and make a combination of these values with each value of the Z variable.
- Fix nominal values of X and Y are 50,40, and we will compare these two values with 79, 80,81,90,99,100 and 101.
- We will fix the nominal values of X and Z and will make a combination of these values with each value of the Y variable.
- Fix nominal values of X and Z are 50, 90, and we will make a combination of these two values with 19, 20,21,40,59,60 and 61.
- We will fix the nominal values of Y and Z and will make a combination of these values with each value of the X variable.
- Fix nominal values of Y and Z are 40, 90, and we will make a combination of these two values with -1, 0,1,50,99,100 and 101.

Test cases generated in Robust simple Boundary Value Testing.

Test Case#	X	Y	Z	Comment
1	50	40	79	Fix Nominal of X and Y
2	50	40	80	Fix Nominal of X and Y
3	50	40	81	Fix Nominal of X and Y
4	50	40	90	Fix Nominal of X and Y
5	50	40	99	Fix Nominal of X and Y
6	50	40	100	Fix Nominal of X and Y
7	50	40	101	Fix Nominal of X and Y
8	50	19	90	Fix Nominal of X and Z
9	50	20	90	Fix Nominal of X and Z
10	50	21	90	Fix Nominal of X and Z
11	50	59	90	Fix Nominal of X and Z
12	50	60	90	Fix Nominal of X and Z
13	50	61	90	Fix Nominal of X and Z
14	-1	40	90	Fix Nominal of Y and Z
15	0	40	90	Fix Nominal of Y and Z
16	1	40	90	Fix Nominal of Y and Z
17	99	40	90	Fix Nominal of Y and Z
18	100	40	90	Fix Nominal of Y and Z
19	101	40	90	Fix Nominal of Y and Z

There are many benefits of robustness testing. Some of the benefits are mentioned below;

1. Better project analysis:

Robustness testing means to increase the study of what has already been analyzed about your product. The robustness testing extends the area of testing of the previously tested software components. Robustness testing also test invalid values to satisfy the testing level.

2. Better design:

The robustness testing result in more options and better software designs and it is completed before the finalization of the design of the product.

3. Achieve consistency:

Robustness testing helps to increase the consistency, reliability, accuracy and efficiency of the software.

5.6 WORST-CASE BOUNDARY VALUE TESTING

Boundary Value analysis uses the critical fault assumption and therefore only tests for a single variable at a time assuming itsextreme values.

By disregarding this assumption, we are able to test the outcome if more than one variable were to assume its extreme value.

In an electronic circuit this is called Worst Case Analysis.

In Worst-Case testing we use this idea to create test cases.

To generate test cases, we take the original 5-tuple set (min, min+, nom, max-, max) and perform the Cartesian product of these values. The end product is a much larger set of results than we have seen before.

The range of x1: 10 to 90

The range of x2: 20 to 70

Testing points detected in Worst Case Boundary Value Testing:

	X1	**X2**
Min	10	20
Min+	11	21
Nominal	50	45
Max-	89	69
Max	90	70

Test cases:

Total test cases = A*A

25 = 5*5

A= Number of testing points.

These testing points are min, min+, nominal, max- and max.

We can generate 25 test cases from both variables x1 and x2 by making a combination of each value of one variable with each value of another variable.

Test cases generated in Worst Case Boundary Value Testing:

Test Case#	**X1,X2**	**Test Case #**	**X1,X2**	**Test Case #**	**X1,X2**
1	10,20	2	10,21	3	10,45
4	10,69	5	10,70	6	11,20
7	11,21	8	11,45	9	11,69
10	11,70	11	50,20	12	50,21
13	50,45	14	50,69	15	50,70
16	89,20	17	89,21	18	89,45
19	89,69	20	89,70	21	90,20
22	90,21	23	90,45	24	90,69
25	90,70				

These 25 test cases are enough test cases to test the input values for these variables.

5.7 SPECIAL VALUE TESTING

Special Value is defined and applied form of Functional Testing, whichis a type of testing that verifies whether each function of the software application operates in conformance with the required specification.

Special value testing is probably the most extensively practiced form of functional testing which is most intuitive and least uniform.

This technique is performed by experienced professionals who are experts in this field and have profound knowledge about the test anddata required for it.

They continuously participate and apply tremendous efforts to deliver appropriate test results to suit the client's requested demand.

WhySpecialValueTesting:

The testing executed by Special Value Testing technique is based on past experiences, which validates that no bugs or defects are left undetected.

The testers are extremely knowledgeable about the industry and use this information while performing Special Value Testing.

Another benefits of opting Special Value Testing technique is that it is ad-hoc in nature

There are no guidelines used by the testers other that their "Best engineering judgment".

The most important aspect of this testing is that, it has had some very valuable inputs and success in finding bugs and errors while testing a software.

5.8 RANDOM TESTING

Random testing is a black-box software testing technique where programs are tested by generating random, independent inputs.

Results of the output are compared against software specifications toverify that the test output is pass or fail.

In case of absence of specifications, the exceptions of the language are used which means if an exception arises during test execution, then it means there is a fault in the program, it is also used as way to avoid biased testing.

Random testing is performed where the defects are NOT identified in regularintervals.

Random input is used to test the system's reliability and performance.

Saves time and effort than actual test efforts.

Monkey Testing:

Monkey testing is a software testing technique in which the testing is performedon the system under test randomly.

In Monkey Testing the tester (sometimes developer too) is considered as the 'Monkey'.

If a monkey uses a computer, he will randomly perform any task on the systemout of his understanding.

Just like the tester will apply random test cases on the system under test to findbugs/errors without predefining any test case.

In some cases, Monkey Testing is dedicated to unit testing

This testing is so random that the tester may not be able to reproduce the error/defect.

The scenario may NOT be definable and may NOT be the correct business case.

Monkey Testing needs testers with very good domain and technical expertise.

The Advantage of Monkey Testing:

As the scenarios that are tested are adhoc, system might be understress so that we can also check for the server responses.

This testing is adopted to complete the testing, in particular if there is a resource/time crunch.

The Disadvantage of Monkey Testing:

No bug can be reproduced: As tester performs tests randomly withrandom data reproducing any bug or error may not be possible.

Less Accuracy: Tester cannot define exact test scenario and even cannot guarantee the accuracy of test cases.

Requires very good technical expertise: It is not worth always to compromise with accuracy, so to make test cases more accuratetesters must have good technical knowledge of the domain.

Fewer bugs and time consuming: This testing can go longer as there isno predefined tests and can find less number of bugs which may cause loopholes in the system.

5.9 GUIDELINES FOR BOUNDARY VALUE ANALYSIS

There are three guidelines for boundary value analysis:

1. One test case for exact boundary values of input domains.

2. One test case for just below boundary value of input domains .

3. One test case for just above boundary values of input domains .

Some examples of Boundary value analysis concept are:

One test case for exact boundary values of input domains each means 1 and 100

One test case for just below boundary value of input domains each means 0 and 99.

One test case for just above boundary values of input domains each means 2 and 101.

For Example: A system can accept the numbers from 1 to 10 numeric values. All other numbers are invalid values. Under this technique, boundary values 0, 1,2, 9,10,11 can be tested.

Another Example is in exam has a pass boundary at 40 percent, merit at 75 percent and distinction at 85 percent. The Valid Boundary values for this scenario will be as follows:

49, 50 - for pass

74, 75 - for merit

84, 85 - for distinction

Boundary values are validated against both the valid boundaries and invalid boundaries.

The Invalid Boundary Cases for the above example can be given as follows:

0 - for lower limit boundary value

101 - for upper limit boundary value

Boundary value analysis is a black box testing and is also applies to white box testing.

Internal data structures like arrays, stacks and queues need to be checked for boundary or limit conditions; when there are linked lists used as internal structures, the behavior of the list at the beginning and end have to be tested thoroughly.

Boundary value analysis help identify the test cases that are most likely to uncover defects.

5.10 EQUIVLANCE CLASS TESTING

Equivalence class testing is better known as Equivalence Class Partitioning and Equivalence Partitioning. This is a renowned testing approach among all other software testing techniques in the market that

allows the testing team to develop and partition the input data for analyzing and testing and based on that the software products are partitioned and divided into number of equivalence classes for testing.

- The equivalence classes that are divided perform the same operation and produce same characteristics or behavior of the inputs provided.
- The test cases are created on the basis on the different attributes of the classes and each input from the each class is used for execution of test cases, validating the software functions and moreover validating the working principles of the software products for the inputs that are given for the respective classes.
- It is also referred as the logical step in functional testing model approach that enhances the quality of test classes and by removing any redundancy or faults that can exist in the testing approach.

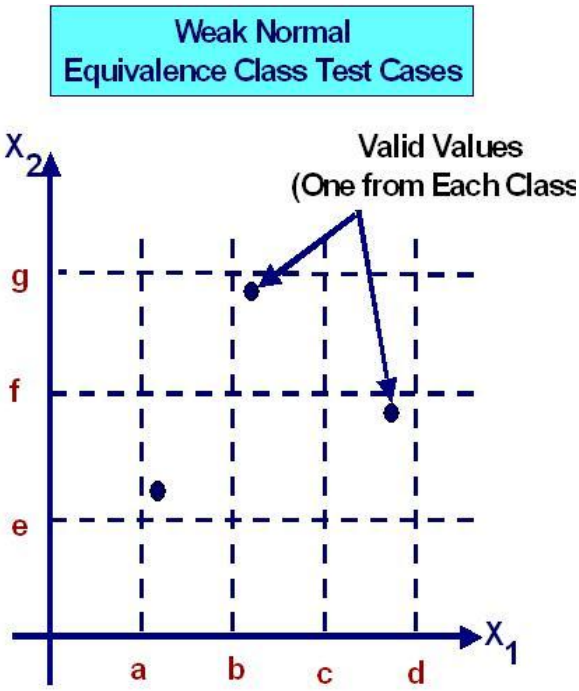
Types of Equivalence Class Testing:

Following four types of equivalence class testing are presented here:

1) Weak Normal Equivalence Class Testing.
2) Strong Normal Equivalence Class Testing.
3) Weak Robust Equivalence Class Testing.
4) Strong Robust Equivalence Class Testing.

1) Weak Normal Equivalence Class Testing:

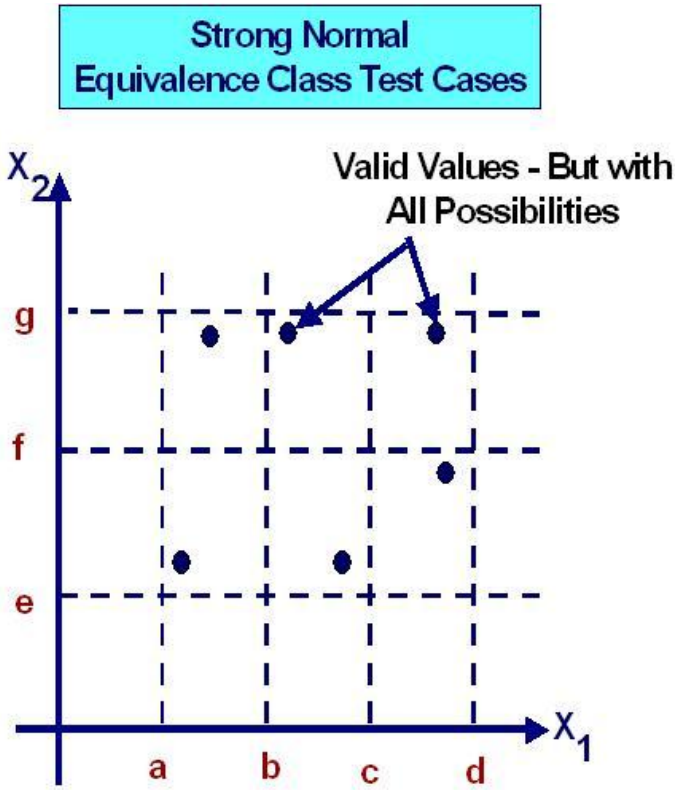
The word 'weak' means 'single fault assumption'. This type of testing is accomplished by using one variable from each equivalence class in a test case. We would, thus, end up with the weak equivalence class test cases as shown in the following figure.



Each dot in above graph indicates a test data. From each class we have one dot meaning that there is one representative element of each test case. In fact, we will have, always, the same number of weak equivalence class test cases as the classes in the partition.

2) Strong Normal Equivalence Class Testing:

This type of testing is based on the multiple fault assumption theory. So, now we need test cases from each element of the Cartesian product of the equivalence classes, as shown in the following figure.



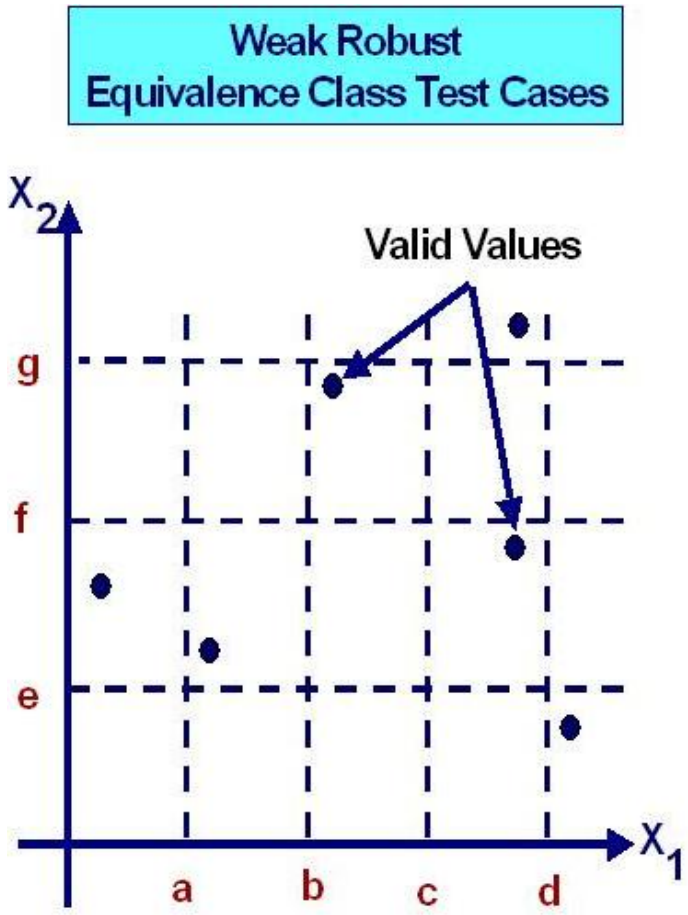
Just like we have truth tables in digital logic, we have similarities between these truth tables and our pattern of test cases. The Cartesian product guarantees that we have a notion of "completeness" in following two ways

a) We cover all equivalence classes.

b) We have one of each possible combination of inputs.

3) Weak Robust Equivalence Class Testing:

The name for this form of testing is counter intuitive and oxymoronic. The word' weak' means single fault assumption theory and the word 'Robust' refers to invalid values. The test cases resulting from this strategy are shown in the following figure.



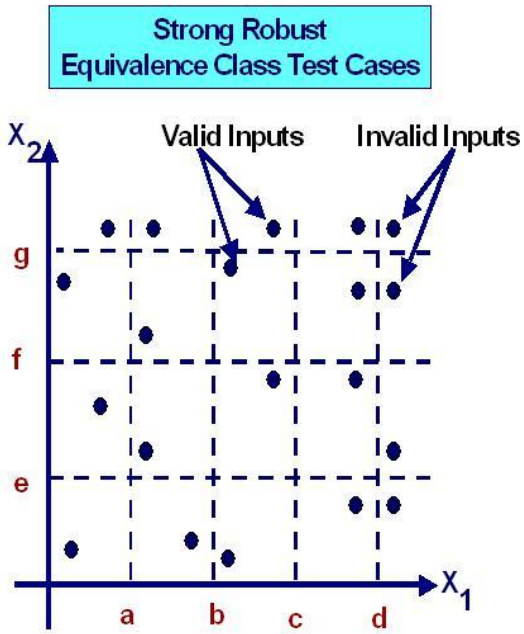
Following two problems occur with robust equivalence testing:

a) Very often the specification does not define what the expected output for an invalid test case should be. Thus, testers spend a lot of time defining expected outputs for these cases.

b) Strongly typed languages like Pascal, Ada, eliminate the need for the consideration of invalid inputs. Traditional equivalence testing is a product of the time when languages such as FORTRAN, C and COBOL were dominant. Thus this type of error was quite common.

4) Strong Robust Equivalence Class Testing:

This form of equivalence class testing is neither counter intuitive nor oxymoronic, but is just redundant. As explained earlier also, 'robust' means consideration of invalid values and the 'strong' means multiple fault assumption. We obtain the test cases from each element of the Cartesian product of all the equivalence classes as shown in the following figure.



We find here that we have 8 robust (invalid) test cases and 12 strong or valid inputs. Each one is represented with a dot. So, totally we have 20 test cases (represented as 20 dots) using this technique.

5.11 TRADITIONAL EQUIVALENCE CLASS TESTING

Programmer arrogance: – in the 1960s and 1970s, programmers often had very detailed input data requirements. – if input data didn't comply, it was the user's fault – the popular phrase—Garbage In, Garbage Out (GIGO)

Programs from this era soon developed defences – (many of these programs are STILL legacy software) – as much as 75% of code verified input formats and values

"Traditional" equivalence class testing focuses on detecting invalidinput. – (almost the same as our "weak robust equivalence class testing")

5.12 IMPROVED EQUIVALENCE CLASS TESTING

There are two main properties that underpin the methods used infunctional testing.

These two properties lead to two different types of equivalence classtesting, weak and strong.

However if one decide to test for invalid i/p or o/p as well as valid i/por o/p we can produce another two different types of equivalence class testing, normal and robust.

Robust equivalence class testing takes into consideration the testing of invalid values, whereas normal does not.

5.13 EDGE TESTING

A hybrid of BVT and Equivalence Class Testing forms the name "Edge Testing."

It is used when contiguous ranges of a particular variable constitute equivalence classes of valid values.

When a programmer makes an error, which results in a defect in the s/w source code.

If this defect is executed, system will produce wrong results, causing a failure.

A defect can be called fault or bug.

Once the set of edge values are determined, edge testing can follow any of the four forms of equivalence class testing

The number of test cases obviously increase as with the variations of BV andECT.

5.14 GUIDELINES FOR EQUIVALENCE CLASS TESTING

The following guidelines are helpful for equivalence class testing:

By following a set of guidelines while implementing the **process of testing**, the team of testers can ensure better outputs from the tests and make sure all scenarios are being tested accurately. Therefore, listed below are some tips/guidelines for equivalence class testing:

- Use robust forms if the error conditions in the software product are of high priority.
- It can be used by the team in projects where the program function is complex.
- To ensure the accuracy and precision of equivalence class testing, define the input data in terms of intervals and sets of discrete values.
- Use of robust from is redundant of the implemented language is strongly types and when invalid values cause runtime errors in the system.
- The team needs to select one valid and one invalid input value each, if the input conditions are broken or not stated accurately.
- Establishing proper equivalence relation might require several tries and extra efforts of the team.

5.15 EQUIVALENCE PARTITIONING EXAMPLE

Example 1 Grocery Store Example:

Consider a software module that is intended to accept the name of a grocery item and a list of the different sizes the item comes in, specified in ounc-es. The specifications state that the item name is to be alphabetic characters 2 to 15 characters in length. Each size may be a value in the range of 1 to 48, whole numbers only. The sizes are to be entered in ascending order (smaller sizes first). A maximum of five sizes may be entered for each item. The item name is to be entered first, followed by a comma, and then followed by a list of sizes. A comma will be used to separate each size. Spaces (blanks) are to be ignored anywhere in the input.

Derived Equivalence Classes:

- Item name is alphabetic (valid)
- Item name is not alphabetic (invalid)
- Item name is less than 2 characters in length (invalid)
- Item name is 2 to 15 characters in length (valid)
- Item name is greater than 15 characters in length (invalid)
- Size value is less than 1 (invalid)
- Size value is in the range 1 to 48 (valid)
- Size value is greater than 48 (invalid)
- Size value is a whole number (valid)
- Size value is a decimal (invalid)
- Size value is numeric (valid)
- Size value includes nonnumeric characters (invalid)
- Size values entered in ascending order (valid)
- Size values entered in no ascending order (invalid)
- No size values entered (invalid)
- One to five size values entered (valid)
- More than five sizes entered (invalid)
- Item name is first (valid)
- Item name is not first (invalid)
- A single comma separates each entry in list (valid)
- A comma does not separate two or more entries in the list (invalid)

- The entry contains no blanks (???)
- The entry contains blanks (????)

Advantages:

1. The ECT requires less effort when performing one test case for one partition
2. Performing test on one test case for each partition consumes less time.
3. Redundancy is removed by eliminating data that yields similar output.

Disadvantages:

1. If there is any mistake in accurately defining any class or if the value selected for finding errors does not properly represent a class, then the errors will be difficult to find.
2. If any class has subclasses, then selecting a value from any one subclass may not represent each subclass.

Example 2 : Equivalence and Boundary Value:

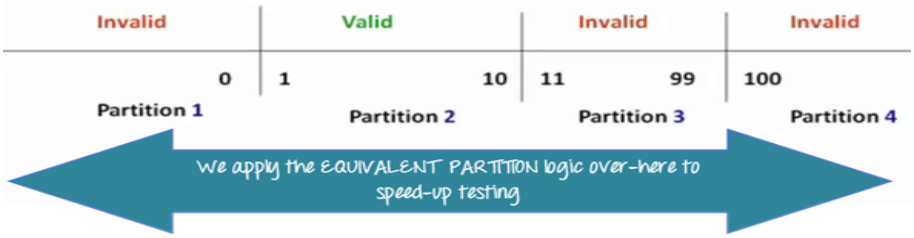
- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Order Pizza:

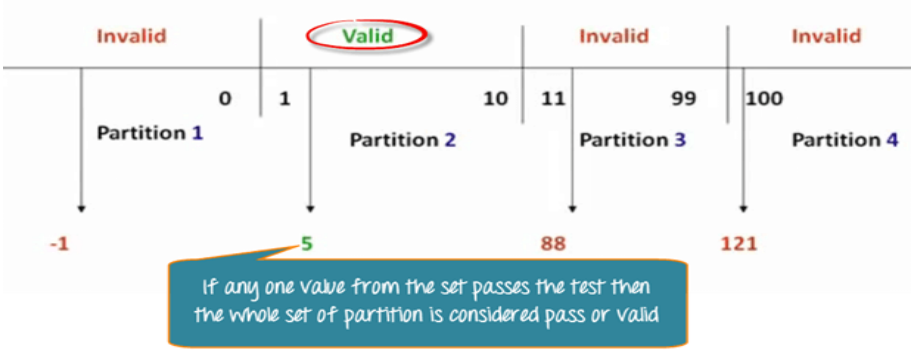
Here is the test condition:

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

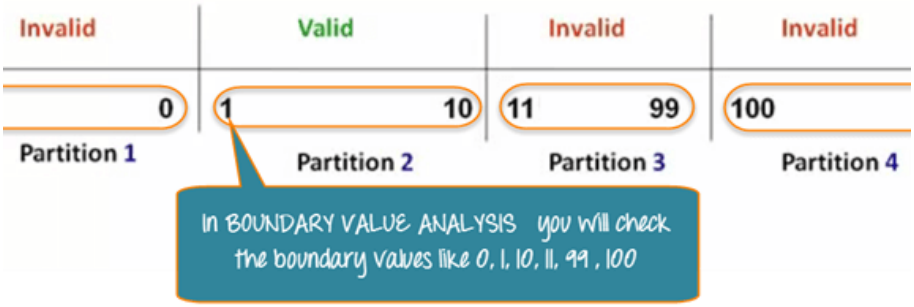
We cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behavior can be considered the same.



The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is that if one condition/value in a partition passes all others will also pass. Likewise, if one condition in a partition fails, all other conditions in that partition will fail.



Boundary Value Analysis: in Boundary Value Analysis, you test boundaries between equivalence partitions



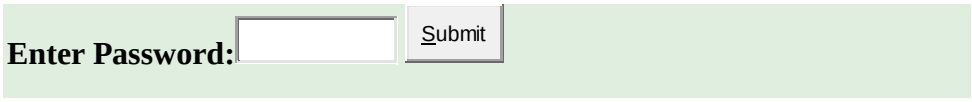
In our earlier equivalence partitioning example, instead of checking one value for each partition, you will check the values at the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at **both valid and invalid boundaries**. Boundary Value Analysis is also called **range checking**.

Equivalence partitioning and boundary value analysis (BVA) are closely related and can be used together at all levels of testing.

Example 3: Equivalence and Boundary Value:

Following password field accepts minimum 6 characters and maximum 10 characters

That means results for values in partitions 0-5, 6-10, 11-14 should be equivalent



Test Scenario #	Test Scenario Description	Expected Outcome
1	Enter 0 to 5 characters in password field	System should not accept
2	Enter 6 to 10 characters in password field	System should accept
3	Enter 11 to 14 character in password field	System should not accept

Examples 3: Input Box should accept the Number 1 to 10:

Here we will see the Boundary Value Test Cases

Test Scenario Description	Expected Outcome
Boundary Value = 0	System should NOT accept
Boundary Value = 1	System should accept
Boundary Value = 2	System should accept
Boundary Value = 9	System should accept
Boundary Value = 10	System should accept
Boundary Value = 11	System should NOT accept

5.16 WHY EQUIVALENCE & BOUNDARY ANALYSIS TESTING

Boundary value analysis and equivalence class testing are two strategies used for test case designing in black box testing, which makes it crucial for us to differentiate them from one another and define their specific relevance in software testing. The differences between these two are:

Equivalence Class Testing	Boundary Value Analysis
1. Equivalence Class Testing is a type of black box technique.	1. Next part of Equivalence Class Partitioning/Testing.
2. It can be applied to any level of testing, like unit, integration, system, and more.	2. Boundary value analysis is usually a part of **stress** & **negative testing**.
3. A **test case design technique** used to divide input data into different equivalence classes.	3. This test case design technique used to test boundary value between partitions.

4. Reduces the time of testing, while using less and effective test cases.	4. Reduces the overall time of test execution, while making **defect detection** faster & easy.
5. Tests only one from each partition of the equivalence classes.	5. Selects test cases from the edges of the equivalence classes.

5.18 SUMMARY

- Boundary Analysis testing is used when practically it is impossible to test a large pool of test cases individually
- Two techniques - Boundary value analysis and equivalence partitioning testing techniques are used
- In Equivalence Partitioning, first, you divide a set of test condition into a partition that can be considered.
- In Boundary Value Analysis you then test boundaries between equivalence partitions
- Appropriate for calculation-intensive applications with variables that represent physical quantities.
- If the range condition is given as an input, then one valid and two invalid equivalence classes are defined.
- If a specific value is given as input, then one valid and two invalid equivalence classes are defined.
- If a member of set is given as an input, then one valid and one invalid equivalence class is defined.
- If Boolean no. is given as an input condition, then one valid and one invalid equivalence class is defined.
- The whole success of equivalence class testing relies on the identification of equivalence classes. The identification of these classes relies on the ability of the testers who creates these classes and the test cases based on them.
- In the case of complex applications, it is very difficult to identify all set of equivalence classes and requires a great deal of expertise from the tester's side.
- Incorrectly identified equivalence classes can lead to lesser test coverage and the possibility of defect leakage.

5.19 EXERCISES

1. Explain Boundary value Analysis?
2. What are important criteria's for Boundary value Analysis?
3. Explain Static and Dynamic Testing?
4. Explain different types of Equivalence classes?

5.20 REFERENCES

- The Art of Software Testing, 3rd Edition Author: Glenford J. Myers, Corey Sandler, Tom Badgett.
- A Practitioner's Guide to Software Test Design Author: Lee Copeland
- Foundations of Software Testing ISTQB Certification
- Software Testing: Principles and Practices pearsons

*****

6

DECISION TABLE BASED TESTING, PATH TESTING, DATA FLOW TESTING

Unit Structure

6.0 OBJECTIVES

- Understand the data flow testing.
- Visualize the decision table
- Understand the need and appreciate the usage of the testing methods.
- Identify the complications in a transaction flow testing method and anomalies in data flow testing.
- Interpret the data flow anomaly state graphs and control flow grpahs and represent the state of the data objetcs.
- Understand the limitations of Static analysis in data flow testing.
- Compare and analyze various strategies of data flow testing.

6.1WHAT IS A DECISION TABLE

It is a table which shows different combination inputs with their associated outputs, this is also known as cause effect table. In EP and BVA we have seen that these techniques can be applied to only specific conditions or inputs however if we have different inputs which result in different actions being taken or in other words we have a business rule to test where there are different combination of inputs which result in different actions.

For testing such rules or logic decision table testing is used.

It is a black box test design technique.

6.1.1 Components of a Decision Table:

Decision table is divided into four parts:

1. Condition
2. Action
3. Stub
4. Entry
5. rules

	R1	R2	R3	R4	R5	R6	R7	R8	
C1	T	T	T	T	F	F	F	F	values of conditions
C2	T	T	F	F	T	T	F	F	
C3	T	F	T	F	T	F	T	F	
a1	x			x	x			x	
a2	x							x	actions taken
a3		x				x			
a4			x	x			x	x	
a5	x			x					

Read a Decision Table by columns of rules : R1 says when all conditions are T, then actions a1, a2, and a5 occur

The conditions in the decision table may take on any number of values.

The decision table allows the iteration of all the combinations of values of the condition, thus it provides a "completeness check."

The conditions in the decision table may be interpreted as the inputs, and the actions may be thought of as outputs. OR conditions needs to be thought as inputs needed set the conditions, and actions can be processing

Consider a program statement that, given the length of 3 sides, determines whether the 3 sides can (i) form a triangle and (ii) what type of triangle (equilateral, isosceles, or scalene).

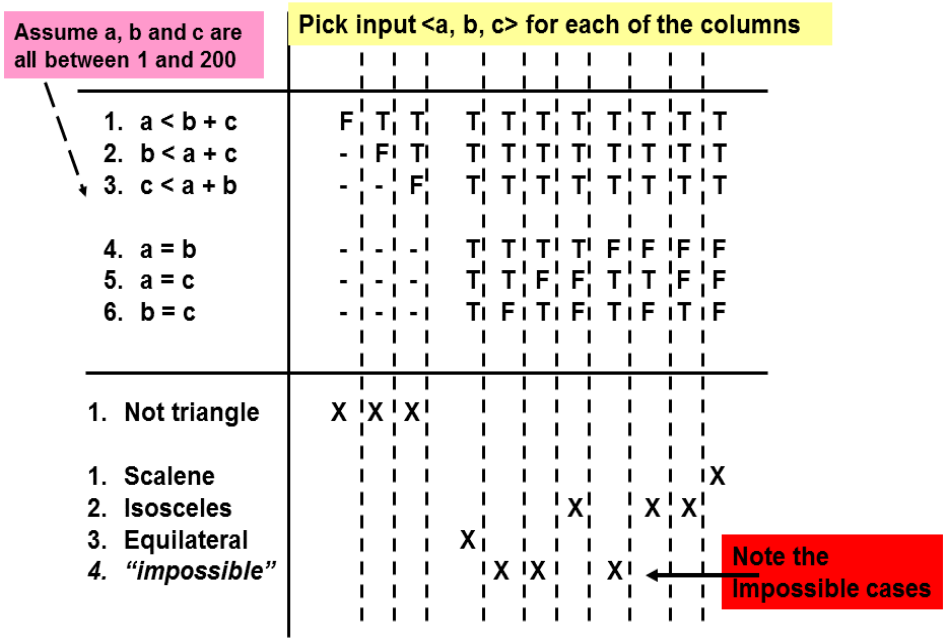
The inputs are a, b, c sides (each between 1 and 200)

Then the inputs must satisfy certain conditions:

a < b + c

b < a + c

c < a + b



Decision table for triangle problem

There is the "invalid situation" --- not a triangle:

There are 3 test conditions in the Decision table

Note the "-" entries, which represents "don't care," when it is determined that the input sides <a, b, c> do not form a triangle

There is the "valid" triangle situation:

There are 3 types of valid; so there are 23 = 8 potential conditions

But there are 3 "impossible" situations

So there are only 8 – 3 = 5 conditions

So, for valid values of a, b, and c, we need to come up with 8 sets of <a, b, c> to test the 8 "Rules".

Decision Table Testing is a good way to deal with combination of inputs, which produce different results

To understand this with an example let's consider the behavior of Flight Button for different combinations of Fly From & Fly To

- When both Fly From & Fly To are not set the Flight Icon is disabled. In the decision table , we register values False for Fly From & Fly To and the outcome would be ,which is Flights Button will be disabled i.e. FALSE
- Next , when Fly From is set but Fly to is not set , Flight button is disabled. Correspondingly you register True for Fly from in the decision table and rest of the entries are false
- When , Fly from is not set but Fly to is set , Flight button is disabled And you make entries in the decision table
- Lastly , only when Fly to and Fly from are set , Flights button is enabled And you make corresponding entry in the decision table
- If you observe the outcomes for Rule 1 , 2 & 3 remain the same .So you can select any of the them and rule 4 for your testing
- The significance of this technique becomes immediately clear as the number of inputs increases. .Number of possible Combinations is given by 2 ^ n , where n is number of Inputs.
- For n = 10 , which is very common is web based testing , having big input forms , the number of combinations will be 1024. Obviously, you cannot test all but you will choose a rich sub-set of the possible combinations using decision based testing

Decision table is based on logical relationships just as the truth table .It is a tool that helps us look at the "complete" combination of conditions technique.

Advantages:

1. The decision- table-based testing works iteratively, which means that if the leading table could not deliver the required result, then the decision –table-based testing helps to develop a new decision table or tables.
2. The decision table assures complete testing.
3. The decision table does not provide any particular order of occurrences of conditions and actions.

Disadvantages:

1. The larger decision tables are required to be divided into smaller tables to reduce redundancy and increase in complexity.
2. The decision tables are not kept proportional.

6.2 WHAT IS STATE TRANSITION TESTING IN SOFTWARE TESTING?

- State transition testing is used where some aspect of the system can be described in what is called a 'finite state machine'. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the 'machine'. This is the model on which the system and the tests are based.
- Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.
- A finite state system is often shown as a **state diagram** (see Figure 4.2).
- One of the advantages of the state transition technique is that the model can be as detailed or as abstract as you need it to be. Where a part of the system is more important (that is, requires more testing) a greater depth of detail can be modeled. Where the system is less important (requires less testing), the model can use a single state to signify what would otherwise be a series of different states.

A state transition model has four basic parts:

- The states that the software may occupy (open/closed or funded/insufficient funds);
- The transitions from one state to another (not all transitions are allowed);
- The events that cause a transition (closing a file or withdrawing money);
- The actions that result from a transition (an error message or being given your cash).

Hence we can see that in any given state, one event can cause only one action, but that the same event – from a different state – may cause a different action and a different end state.

For example, if you request to withdraw $100 from a bank ATM, you may be given cash. Later you may make exactly the same request but it may refuse to give you the money because of your insufficient balance. This later refusal is because the state of your bank account has changed from having sufficient funds to cover the withdrawal to having insufficient funds. The transaction that caused your account to change its state was probably the earlier withdrawal. A state diagram can represent a model from the point of view of the system, the account or the customer.

Let us consider another example of a word processor. If a document is open, you are able to close it. If no document is open, then 'Close' is not available. After you choose 'Close' once, you cannot choose it again for

the same document unless you open that document. A document thus has two states: open and closed.

We will look first at test cases that execute valid state transitions. Figure 4.2 below, shows an example of entering a Personal Identity Number (PIN) to a bank account. The states are shown as circles, the transitions as lines with arrows and the events as the text near the transitions. (We have not shown the actions explicitly on this diagram, but they would be a message to the customer saying things such as 'Please enter your PIN'.)

The state diagram shows seven states but only four possible events (Card inserted, Enter PIN, PIN OK and PIN not OK). We have not specified all of the possible transitions here – there would also be a time-out from 'wait for PIN' and from the three tries which would go back to the start state after the time had elapsed and would probably eject the card. There would also be a transition from the 'eat card' state back to the start state. We have not specified all the possible events either – there would be a 'cancel' option from 'wait for PIN' and from the three tries, which would also go back to the start state and eject the card.

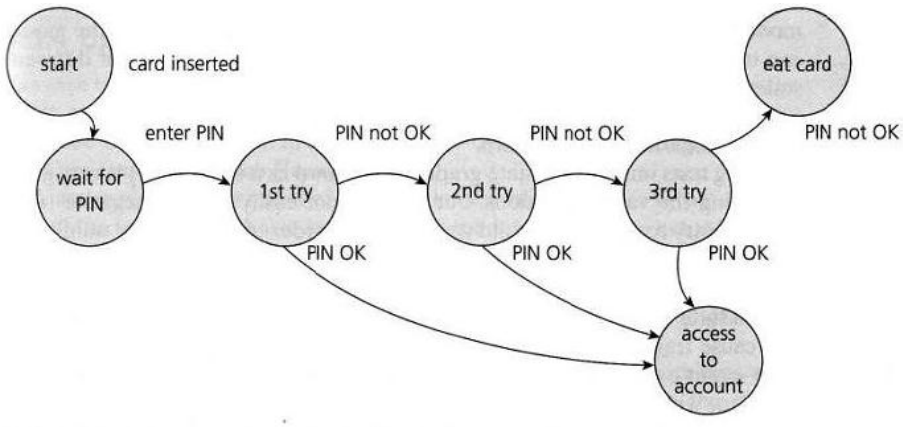


FIGURE 4.2 State diagram for PIN entry

In deriving test cases, we may start with a typical scenario:

- First test case here would be the normal situation, where the correct PIN is entered the first time.
- A second test (to visit every state) would be to enter an incorrect PIN each time, so that the system eats the card.
- A third test we can do where the PIN was incorrect the first time but OK the second time, and another test where the PIN was correct on the third try. These tests are probably less important than the first two.
- Note that a transition does not need to change to a different state (although all of the transitions shown above do go to a different state). So there could be a transition from 'access account' which just goes back to 'access account' for an action such as 'request balance'.

Test conditions can be derived from the state graph in various ways. Each state can be noted as a test condition, as can each transition. However this state diagram, even though it is incomplete, still gives us information on which to design some useful tests and to explain the state transition technique.

6.3 WHAT IS USE CASE TESTING IN SOFTWARE TESTING?

- Use case testing is a technique that helps us identify test cases that exercise the whole system on a transaction by transaction basis from start to finish. They are described by Ivar Jacobson in his book Object-Oriented Software Engineering: A Use Case Driven Approach [Jacobson, 1992].
- A use case is a description of a particular use of the system by an actor (a user of the system). Each use case describes the interactions the actor has with the system in order to achieve a specific task (or, at least, produce something of value to the user).
- Actors are generally people but they may also be other systems.
- Use cases are a sequence of steps that describe the interactions between the actor and the system. Use cases are defined in terms of the actor, not the system, describing what the actor does and what the actor sees rather than what inputs the system expects and what the system's outputs.
- They often use the language and terms of the business rather than technical terms, especially when the actor is a business user.
- They serve as the foundation for developing test cases mostly at the system and acceptance testing levels.
- Use cases can uncover integration defects, that is, defects caused by the incorrect interaction between different components. Used in this way, the actor may be something that the system interfaces to such as a communication link or sub-system.
- Use cases describe the process flows through a system based on its most likely use. This makes the test cases derived from use cases particularly good for finding defects in the real-world use of the system (i.e. the defects that the users are most likely to come across when first using the system).
- Each use case usually has a mainstream (or most likely) scenario and sometimes additional alternative branches (covering, for example, special cases or exceptional conditions).
- Each use case must specify any preconditions that need to be met for the use case to work.

- Use cases must also specify post conditions that are observable results and a description of the final state of the system after the use case has been executed successfully.

The ATM PIN example is shown below in Figure 4.3. We show successful and unsuccessful scenarios. In this diagram we can see the interactions between the A (actor – in this case it is a human being) and S (system). From step 1 to step 5 that is success scenario it shows that the card and pin both got validated and allows Actor to access the account. But in extensions there can be three other cases that is 2a, 4a, 4b which is shown in the diagram below.

For use case testing, we would have a test of the success scenario and one testing for each extension. In this example, we may give extension 4b a higher priority than 4a from a security point of view.

System requirements can also be specified as a set of use cases. This approach can make it easier to involve the users in the requirements gathering and definition process.

	Step	Description
Main Success Scenario **A: Actor** **S: System**	1	A: Inserts card
	2	S: Validates card and asks for PIN
	3	A: Enters PIN
	4	S: Validates PIN
	5	S: Allows access to account
Extensions	2a	Card not valid S: Display message and reject card
	4a	PIN not valid S: Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S: Eat card and exit

6.4 PATH TESTING

Basis Path Testing is a white-box testing technique based on the control structure of a program or a module. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing. Therefore, by definition,

Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module.

Path Testing is a method that is used to design the test cases. In path testing method, the control flow graph of a program is designed to find a set of linearly independent paths of execution. In this method Cyclomatic

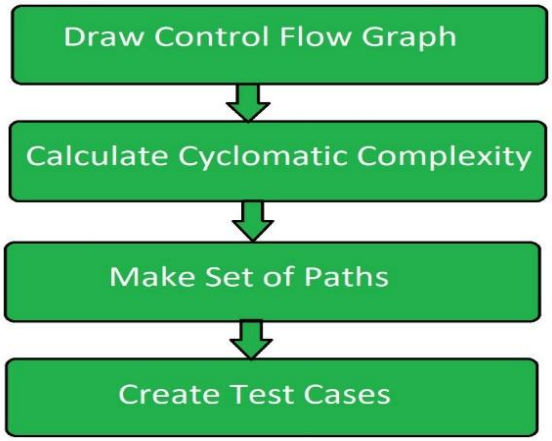
Complexity is used to determine the number of linearly independent paths and then test cases are generated for each path.

It give complete branch coverage but achieves that without covering all possible paths of the control flow graph. McCabe's Cyclomatic Complexity is used in path testing. It is a structural testing method that uses the source code of a program to find every possible executable path.

Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure. To design test cases using this technique, four steps are followed:

1. Construct the Control Flow Graph
2. Compute the Cyclomatic Complexity of the Graph
3. Identify the Independent Paths
4. Design Test cases from Independent Paths

6.4.1 Path Testing Process:

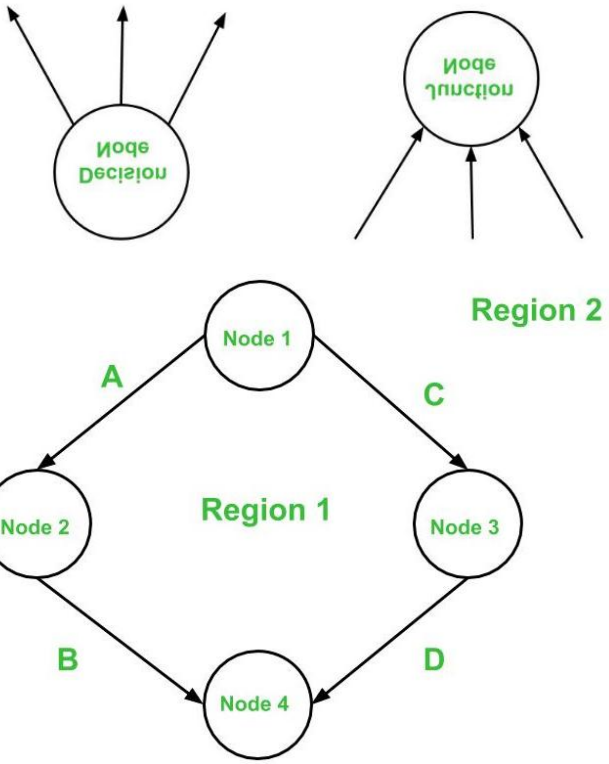


Let's understand each step one by one.

1. Control Flow Graph:

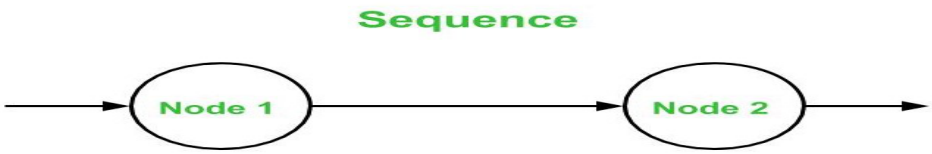
A control flow graph (or simply, flow graph) is a directed graph which represents the control structure of a program or module. A control flow graph (V, E) has V number of nodes/vertices and E number of edges in it. A control graph can also have :

- **Junction Node:** a node with more than one arrow entering it.
- **Decision Node:** a node with more then one arrow leaving it.
- **Region:** area bounded by edges and nodes (area outside the graph is also counted as a region.).

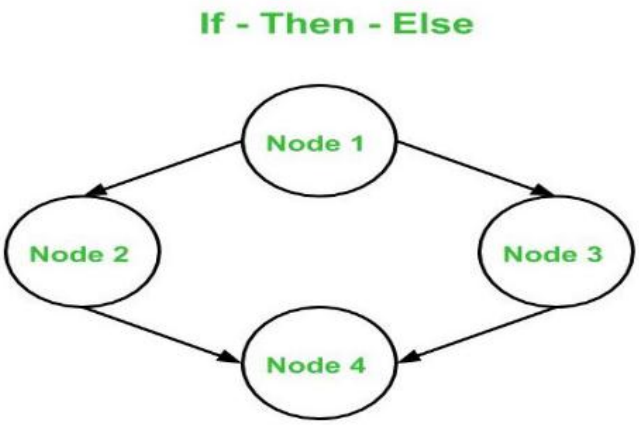


Below are the **notations** used while constructing a flow graph:

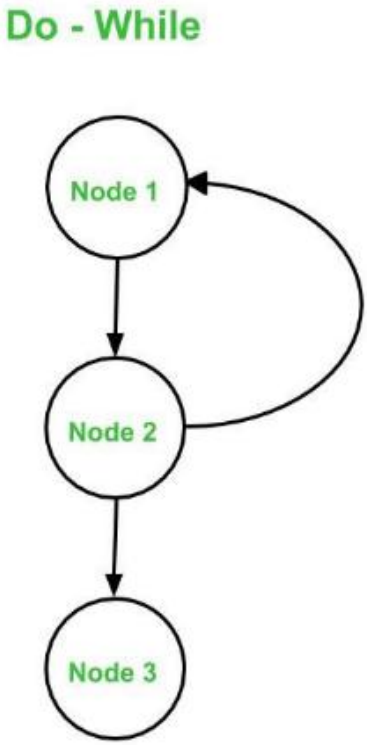
Sequential Statements:



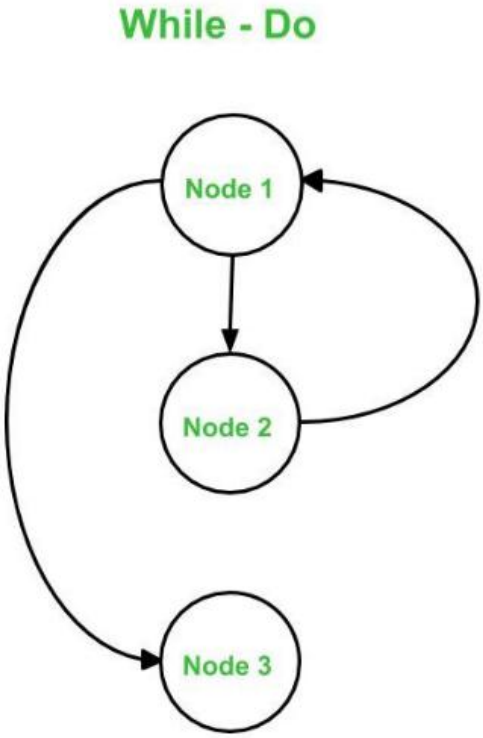
If – Then – Else –



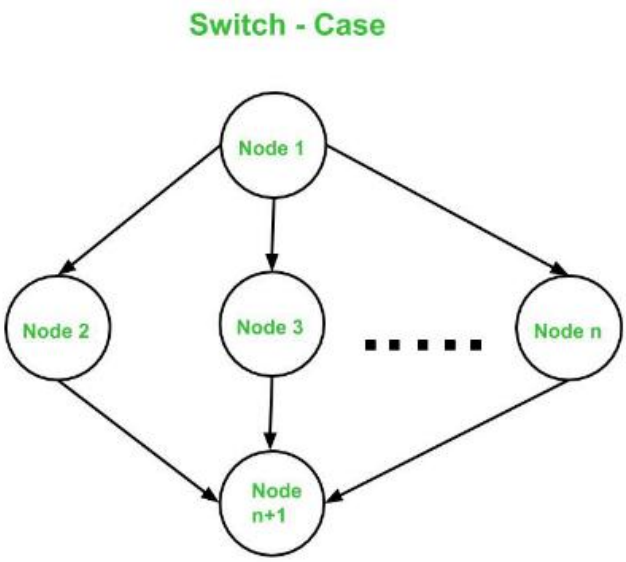
Do – While –



While – Do



Switch – Case –



6.4.2 Cyclomatic Complexity:

The cyclomatic complexity V(G) is said to be measure of the logical complexity of a program. It can be calculated using three different formulae:

Formula based on edges and nodes:

V(G) = e - n + 2*P

Where,
e is number of edges,

n is number of vertices,

P is number of connected components.

For example, consider first graph given above,

where, e = 4, n = 4 and p = 1

where, e = 4, n = 4 and p = 1

So,

Cyclomatic complexity V(G)

= 4 - 4 + 2 * 1

= 2

Formula based on Decision Nodes:

V(G) = d + P

where,
d is number of decision nodes,

P is number of connected nodes.

For example, consider first graph given above,

where, d = 1 and p = 1

1. So,
2. Cyclomatic Complexity V(G)
3. = 1 + 1
4. = 2
5. **Formula based on Regions:**

V(G) = number of regions in the graph

For example, consider first graph given above,

1. Cyclomatic complexity V(G)
2. = 1 (for Region 1) + 1 (for Region 2)
3. = 2

Hence, using all the three above formulae, the cyclomatic complexity obtained remains same. All these three formulae can be used to compute and verify the cyclomatic complexity of the flow graph.

Note:

1. For one function [e.g. Main( ) or Factorial( )], only one flow graph is constructed. If in a program, there are multiple functions, then a separate flow graph is constructed for each one of them. Also, in the cyclomatic complexity formula, the value of 'p' is set depending of the number of graphs present in total.
2. If a decision node has exactly two arrows leaving it, then it is counted as one decision node. However, if there are more than 2 arrows leaving a decision node, it is computed using this formula :

$$d = k - 1$$

Here, k is number of arrows leaving the decision node.

6.4.3 Independent Paths:

An independent path in the control flow graph is the one which introduces at least one new edge that has not been traversed before the path is defined. The cyclomatic complexity gives the number of independent paths present in a flow graph. This is because the cyclomatic complexity is used as an upper-bound for the number of tests

that should be executed in order to make sure that all the statements in the program have been executed at least once.

Consider first graph given above here the independent paths would be 2 because number of independent paths is equal to the cyclomatic complexity.

So, the independent paths in above first given graph:

- **Path 1:**

A -> B

- **Path 2:**

C -> D

Note:

Independent paths are not unique. In other words, if for a graph the cyclomatic complexity comes out be N, then there is a possibility of obtaining two different sets of paths which are independent in nature.

6.4.4 Design Test Cases:

Finally, after obtaining the independent paths, test cases can be designed where each test case represents one or more independent paths.

Advantages:

Basis Path Testing can be applicable in the following cases:

1. **More Coverage:**

Basis path testing provides the best code coverage as it aims to achieve maximum logic coverage instead of maximum path coverage. This results in an overall thorough testing of the code.

2. **Maintenance Testing:**

When a software is modified, it is still necessary to test the changes made in the software which as a result, requires path testing.

3. **Unit Testing:**

When a developer writes the code, he or she tests the structure of the program or module themselves first. This is why basis path testing requires enough knowledge about the structure of the code.

4. **Integration Testing:**

When one module calls other modules, there are high chances of Interface errors. In order to avoid the case of such errors, path testing is performed to test all the paths on the interfaces of the modules.

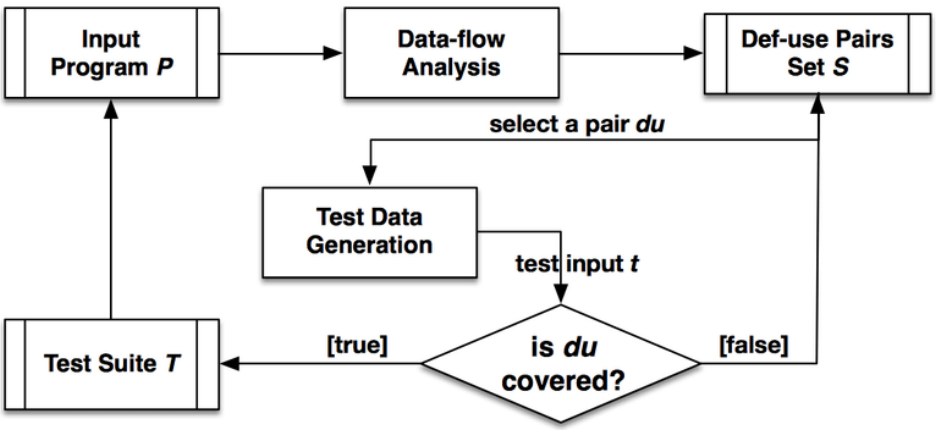
5. Testing Effort:

Since the basis path testing technique takes into account the complexity of the software (i.e., program or module) while computing the cyclomatic complexity, therefore it is intuitive to note that testing effort in case of basis path testing is directly proportional to the complexity of the software or program.

6.5 DATA FLOW TESTING

Data Flow Testing is a specific strategy of software testing that focuses on data variables and their values. It makes use of the control flow graph. When it comes to categorization Data flow testing will can be considered as a type of white box testing and structural types of testing. It keeps a check at the data receiving points by the variables and its usage points. It is done to cover the path testing and branch testing gap. The process is conducted to detect the bugs because of the incorrect usage of data variables or data values. For e.g. Initialization of data variables in programming code, etc.

Data flow testing is a white-box testing technique that examines the data flow with respect to the variables used in the code. It examines the initialization of variables and checks their values at each instance.



6.5.1 Types of data flow testing:

There are two types of data flow testing:

Static data flow testing:

The declaration, usage, and deletion of the variables are examined without executing the code. A control flow graph is helpful in this.

Dynamic data flow testing:

The variables and data flow are examined with the execution of the code.

What is Data flow Testing?

- The programmer can perform numerous tests on data values and variables. This type of testing is referred to as data flow testing.
- It is performed at two abstract levels: static data flow testing and dynamic data flow testing.
- The static data flow testing process involves analyzing the source code without executing it.
- Static data flow testing exposes possible defects known as data flow anomaly.
- Dynamic data flow identifies program paths from source code.

Let us understand this with the help of an example.

1. read x;	
2. If(x>0)	(1, (2, t), x), (1, (2, f), x)
3. a= x+1	(1, 3, x)
4. if (x<=0) {	(1, (4, t), x), (1, (4, f), x)
5. if (x<1)	(1, (5, t), x), (1, (5, f), x)
6. x=x+1; (go to 5)	(1, 6, x)
else	
7. a=x+1	(1, 7, x)
8. print a;	(6,(5, f)x), (6,(5,t)x)
	(6, 6, x)
	(3, 8, a), (7, 8, a).

There are 8 statements in this code. In this code we cannot cover all 8 statements in a single path as if 2 is valid then **4, 5, 6, 7** are not traversed, and if 4 is valid then statement 2 and 3 will not be traversed. Hence we will consider two paths so that we can cover all the statements.

x= 1

Path – 1, 2, 3, 8

Output = 2

If we consider **x = 1**, in step 1; x is assigned a value of 1 then we move to step 2 (since, x>0 we will move to statement **3 (a= x+1)** and at end, it will go to statement 8 and print x =2.

For the second path, we assign x as 1

Set x= -1

Path = 1, 2, 4, 5, 6, 5, 6, 5, 7, 8

Output = 2

X is set as 1 then it goes to step 1 to assign x as 1 and then moves to step 2 which is false as x is smaller than 0 (x>0 and here x=-1). It will then move to step 3 and then jump to step 4; as 4 is true (x<=0 and their x is less than 0) it will jump on 5 (x<1) which is true and it will move to step 6 **(x=x+1)** and here x is increased by 1.

So,

x=-1+1

x=0

x become 0 and it goes to step 5(x<1),as it is true it will jump to step

6 (x=x+1)

x=x+1

x= 0+1

x=1

x is now 1 and jump to step 5 (x<1) and now the condition is false and it will jump to step 7 **(a=x+1)** and set a=2 as x is 1. At the end the value of a is 2. And on step 8 we get the output as 2.

6.5.2 Steps of Data Flow Testing:

- Creation of a data flow graph.
- Selecting the testing criteria.
- Classifying paths that satisfy the selection criteria in the data flow graph.
- Develop path predicate expressions to derive test input.

The life cycle of data in programming code:

Definition:

It includes defining, creation and initialization of data variables and the allocation of the memory to its data object.

Usage:

It refers to the user of the data variable in the code. Data can be used in two types as a predicate (P) or in the computational form(C).

Deletion:

Deletion of the Memory allocated to the variables.

6.5.3Types of Data Flow Testing:

Static Data Flow Testing:

No actual execution of the code is carried out in Static Data Flow testing. Generally, the definition, usage and kill pattern of the data variables is scrutinized through a control flow graph.

Dynamic Data Flow Testing:

The code is executed to observe the transitional results. Dynamic data flow testing includes:

Identification of definition and usage of data variables.

Identifying viable paths between definition and usage pairs of data variables. Designing & crafting test cases for these paths.

Advantages of Data Flow Testing:

- Variables used but never defined,
- Variables defined but never used,
- Variables defined multiple times before actually used,
- DE allocating variables before using.

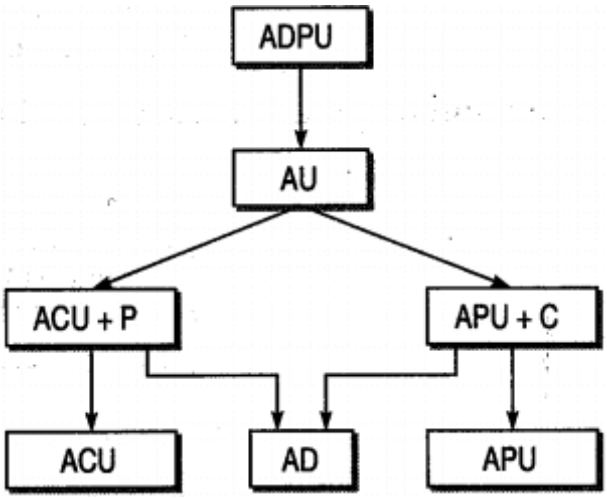
Data Flow Testing Limitations

- Testers require good knowledge of programming.
- Time-consuming
- Costly process.

6.5.4 Data Flow Testing Coverage:

- **All definition coverage:** Covers “sub-paths” from each definition to some of their respective use.
- **All definition-C use coverage:** “sub-paths” from each definition to all their respective C use.
- **All definition-P use coverage:** “sub-paths” from each definition to all their respective P use.
- **All use coverage:** Coverage of “sub-paths” from each definition to every respective use irrespective of types.
- **All definition use coverage:** Coverage of “simple sub-paths” from each definition to every respective use.

6.5.5 Data Flow Testing Strategies:



Following are the test selection criteria:

1. **All-defs:** For every variable x and node i in a way that x has a global declaration in node I, pick a comprehensive path including the def-clear path from node i to Edge (j,k) having a p-use of x or Node j having a global c-use of x

2. **All c-uses:** For every variable x and node i in a way that x has a global declaration in node i, pick a comprehensive path including the def-clear path from node i to all nodes j having a global c-use of x in j.

3. **All p-uses:** For every variable x and node i in a way that x has a global declaration in node i, pick a comprehensive path including the def-clear path from node i to all edges (j,k) having p-use of x on edge (j,k).

4. **All p-uses/Some c-uses:** it is similar to all p-uses criterion except when variable x has no global p-use, it reduces to some c-uses criterion as given below

5. **Some c-uses:** For every variable x and node i in a way that x has a global declaration in node i, pick a comprehensive path including the def-clear path from node i to some nodes j having a global c-use of x in node j.

6. **All c-uses/Some p-uses:** It is similar to all c-uses criterion except when variable x has no global c-use, it reduces to some p-uses criterion as given below:

7. **Some p-uses:** For every variable x and node i in a way that x has a global declaration in node i, pick a comprehensive path including def-clear paths from node i to some edges (j,k) having a p-use of x on edge (j,k).

8. **All uses:** it is a combination of all p-uses criterion and all c-uses criterion.

9. **All du-paths:** For every variable x and node i in a way that x has a global declaration in node i, pick a comprehensive path including all du-paths from node i

- To all nodes j having a global c-use of x in j and
- To all edges (j,k) having a p-use of x on (j,k).

6.5 DATA FLOW TESTING APPLICATIONS

As per studies defects identified by executing 90% "data coverage" is twice as compared to bugs detected by 90% branch coverage. The process flow testing is found effective, even when it is not supported by automation.

It requires extra record keeping; tracking the variables status. The computers help easy tracking of these variables and hence reducing the testing efforts considerably. Data flow testing tools can also be integrated into compilers.

6.6 CONCLUSION

- Data is a very important part of software engineering. The testing performed on data and variables play an important role in software engineering. Hence this is a very important part and should be properly carried out to ensure the best working of your product.
- In path testing method, the control flow graph of a program is designed to find a set of linearly independent paths of execution.
- McCabe's Cyclomatic Complexity is used in path testing. It is a structural testing method that uses the source code of a program to find every possible executable path.
- Decision table testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form.
- Decision table helps to check all possible combinations of conditions for testing and testers can also identify missed conditions easily. The conditions are indicated as True(T) and False(F) values.
- Decision Table Testing is Important because it helps to test different combinations of conditions and provide better test coverage for complex business logic.

6.7 EXERCISE

1. Explain Decision table?
2. How to calculate Cyclomatic Complexity?
3. What are independent Paths?
4. What is static and Dynamic Testing?

6.8 REFERENCES

- The Art of Software Testing, 3rd Edition Author: Glenford J. Myers, Corey Sandler, Tom Badgett.
- A Practitioner's Guide to Software Test Design Author: Lee Copeland
- Foundations of Software Testing ISTQB Certification
- Software Testing: Principles and Practices pearsons
- www.guru.com
- www.istbq.com
- www.tutorial.com

*****

UNIT - IV

7

SOFTWARE VERIFICATION AND VALIDATION

Unit Structure

7.0 OBJECTIVES

After going through this chapter, you will be able to:

- Verification and its Workbench
- Methods of Verification
- Types of Review
- Validation and its Workbench
- Levels of Validation

7.1 INTRODUCTION

Verification and validation (V & V) have become important, especially in software, as the complexity of software in systems has increased, and planning for V & V is necessary from the beginning of the development life cycle. Over the past 20 to 30 years, software development has evolved from small tasks involving a few people to enormously large tasks involving many people. Because of this change, verification and validation has similarly also undergone a change. Previously, verification and validation was an informal process performed by the software engineer himself. However, as the complexity of systems increased, it became obvious that continuing this type of testing would result in unreliable products. It became necessary to look at V & V as a separate activity in the overall software development life cycle.

7.2 VERIFICATION

Verification is the process of evaluating work-products of a development phase to determine whether they meet the specified requirements.

- Verification ensures that the product is built according to the requirements and design specifications.
- It also answers the question, **Are we building the product rightly?**
- Verification is also called "Static technique" or "Conformance to Requirements" as it does not involve execution of any code, program or work product.

7.2.1 Advantages of Verification:

- Verification helps in lowering down the count of the defect in the later stages of development.
- Verifying the product at the starting phase of the development will help in understanding the product in a better way.
- It reduces the chances of failures in the software application or product.
- It helps in building the product as per the customer specifications and needs.
- It helps to reduce the cost of finding and fixing defects. Sometimes defects are fixed as and when they are found.
- Cost of fixing defects is less as there is no impact on other parts of the software.

7.2.2 Disadvantages of Verification:

- It does not show whether the product is correct or not. It shows only whether the process is correctly followed or not.

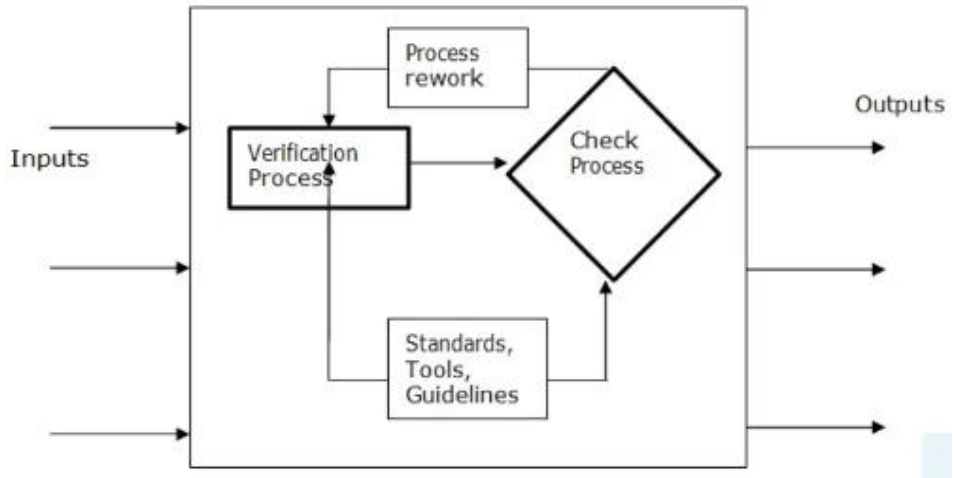
- If the processes are not good, then the outcome may not be good.
- It does not cover any kind of execution of work products.
- "Fit for Use" cannot be assessed in verification.

7.2.3 Prerequisites For Verification:

- Training Required by people for conducting verification
- Standards, guidelines and tools to be used during verification
- Verification Do and Check process definition.

7.3 WORKBENCH FOR VERIFICATION

Verification workbench is where verification activities are conducted either physical or virtual. For every workbench following basic things are required.



Input:

It is the initial workbench stage. Each certain assignment should contain its initial and outcome (input and output) requirements to know the available parameters and expected results. Each workbench has its specific inputs depending on the type of product under testing.

Verification Process:

It describes step by step activities to be conducted in a workbench. It must also describe the activities done while verifying the product under review.

Check:

It is an examination of output parameters after the performance phase to verify its accordance with the expected ones.

Production output:

It is the final stage of a workbench in case the check confirmed the properly conducted performance.

Reworking:

If the outcome parameters are not in compliance with the desired result, it is necessary to return to the verification process and conduct it from the beginning.

7.4 METHODS OF VERIFICATION.

There are many methods available for verification of software product. Some of them are

A. Self Review:

1. Self Review may not be referred to as an official review.
2. Everybody does a self check before giving a work product for further verification.
3. One must capture the self review records and defect found in self review to improve the process.
4. It is a self learning and retrospection process.

Advantages of Self Review

1. Self Reviews are highly flexible with respect to time and defect finding. It may be an online activity.
2. Self Review is an excellent tool for self learning.
3. There is no ego involved in self review can help in self education and self improvement.

Disadvantages of Self Review:

1. Approach or understanding related defect may not be found in self review
2. People involved in self review may not conduct a review in reality due to time and focus issues.

B. Peer Review:

The very easiest method and informal way of reviewing the documents or the programs/software for the purpose of finding out the faults during the verification process is the peer-review method. In this method, we give the document or software programs to others and ask them to review those documents or software programs where we expect their views about the quality of our product and also expect them to find the faults in the program/document.

Online Peer Review:

In this review author and reviewer meet together and review the work jointly. Any explanation required by the reviewer may be provided by the author. The defects are found and corrected jointly by the peers.

Offline Peer Review:

In such kind of review the author informs the reviewer that the work product is ready for the review. The reviewer reviews the work product as per the time availability. The review report is sent to the author along with the defects then the author may decide to accept or reject it.

Advantages of Peer Review:

1. It is informal and unplanned. It can happen at any time.
2. There is no or less ego or pride attached with the review.
3. As defects are discussed and decisions are taken informally defects can be fixed fast.

Disadvantages of Peer Review:

1. The other person doing peer review may not be expert on the artifacts under review so suggestions may not always be valid.
2. Peer may fix the defects without recording them. This may happen in offline review.
3. Sometimes peer may change the defect without informing the author.

C. Walkthrough:

Walkthrough is more formal than peer review but less formal than inspection. It can be called as semi formal review. In typical walkthrough some members of the project team are involved in examining an artifact under review. In a walkthrough, the author of the software document presents the document to other persons which can range from 2 to 7. Participants are not expected to prepare anything. The presenter is responsible for preparing the meeting. The document(s) is/are distributed to all participants. At the time of the meeting of the walk-through, the author introduces the content in order to make them familiar with it and all the participants are free to ask their doubts.

Advantages:

1. Walkthrough is useful for making joint decisions and each member must be involved in making decisions.
2. Defects are recorded and suggestions can be received from the team for improving the work product.

Disadvantages:

1. Availability of people can be an issue when teams are large.
2. Time can be a constraint
3. Members in the team may not be expert in giving comments, so may need some training and basic knowledge about the project.

D. Inspection (FORMAL REVIEW):

1. It is the most formal review type
2. It is led by the trained moderators
3. During inspection the documents are prepared and checked thoroughly by the reviewers before the meeting
4. It involves peers to examine the product
5. A separate preparation is carried out during which the product is examined and the defects are found
6. The defects found are documented in a logging list or issue log
7. A formal follow-up is carried out by the moderator applying exit criteria

Advantages of Inspection:

1. Helps in the Early removal of major defects.
2. This inspection enables a numeric quality assessment of any technical document.
3. Software inspection helps in process improvement.
4. It helps in staff training on the job.
5. Software inspection helps in gradual productivity improvement.

Disadvantages of Software Inspection:

1. It is a time-consuming process.
2. Software inspection requires discipline.
3. Expert opinion may vary from realities as it may be derived from judgements and experiences.

Phases of Inspection:

Planning:

The planning phase starts when the entry criteria for the inspection state are met. The moderator planned the inspection. A moderator verifies that the product entry criteria are met.

Kick-off Inspection:

In the overview phase, a presentation is given to the inspector with some background information needed to review the software product properly. Here objective of the inspection is explained and process to be followed.

Preparation:

This is considered an individual activity. In this part of the process, the inspector collects all the materials needed for inspection, reviews that material, and notes any defects.

Meeting:

The moderator conducts the meeting. In the meeting, the defects are collected and reviewed within a defined time frame.

Rework:

The author performs this part of the process in response to defect disposition determined at the meeting.

Follow-up:

In follow-up, the moderator makes the corrections and then compiles the inspection management and defects summary report. The findings can be used be used to gather statistics about work product, project and progress.

Roles and Responsibilities:

Author:

The author is the person who wrote the document being inspected. He or she is present at the inspection to answer questions to help others understand the work, but not to "defend" his or her work.

Moderator:

The moderator runs the inspection and enforces the protocols of the meeting. The moderator's job is mainly one of controlling interactions and keeping the group focused on the purpose of the meeting—to discover (but not fix) deficiencies. The moderator also ensures that the group does not go off on tangents and sticks to a schedule.

Reader:

The reader calls attention to each part of the document in turn, and thus paces the inspection.

Recorder:

Whenever any problem is uncovered in the document being inspected, the recorder describes the defect in writing. After the inspection, the recorder and moderators prepare the inspection report.

Inspectors:

Inspectors raise questions, suggest problems, and criticize the document. Inspectors are not supposed to "attack" the author or the document but should be objective and constructive. Everyone except the author can act as an inspector.

E. Audit:

Audit means an independent examination of a software product or processes to assess compliance with specifications, standards, contractual agreements, or other criteria. Software Development and testing process audit is an examination of product to ensure that the product as well as the process used to build them predefined criteria. The outcome of the audit report comprising the following.

Major and Minor Non-Conformance:

Non conformance is the deviation between what is required and what actually exists.

- **Minor Non-Conformance:** Minor Non-Conformance indicates that there is some deviation at some place and at other places the process is followed correctly.
- **Major Non-Conformance:** Major Non-Conformance indicates a big issue where something required is missing completely.

Observations:

Observations are findings which may be converted into future non conformances if proper care is nottaken. They may be termed as conformance just on the verge of breakage.

Achievement or Good Practices:

These are good achievements by areas under audit which can be used by others.

In software development and testing phases, various audit are conducted, as mentioned below

A. Kick Off Audit:

This audit covers the areas that ensure whether all the processes required at the start of the project are covered or not. It may start from proposal, contract, risk analysis, scope definition, team size and Skill requirements, authorities and responsibilities of various roles in the team. Kick-off audits cover checking documentation and compliances required at the start of the project.

B. Periodic Software Quality Assurance Audit:

Software quality assurance audit is famous by the name 'SQA audit' or 'SQA review'. It is conducted for a product under development and processes used as defined by quality assurance process definition for these work products. Auditor is expected to check whether different work products meet the defined exit criteria or not, and the process used for building these work products are correctly implemented or not.

C. Phase-End Audit:

Phase-end audit checks whether the phase defined in the development life cycle achieves its expected outcome or not. It is also used as a gate to decide whether the next phase can be started or not. These are also termed **'gate reviews'.**

D. Pre Delivery Audit:

Pre Delivery audit checks whether all the requisite processes of delivery are followed or not, and whether the work product meets the expected delivery criteria or not. Only those work products which are successful in pre delivery audits can be given to a customer.

E. Product Audit:

Product audits can be Covered in SQA audit. It is done by executing sample test cases to find whether the product meets its defined exit criteria or not. Sometimes, this is also considered as 'smoke testing'.

7.5 TYPES OF REVIEWS ON THE BASIS OF STAGE PHASE

In-Process Review:

Reviews done while different phases of software development life cycle are going on are defined as '**In-process review**'. They are intended to check whether inputs to the phase are correct or not, and whether all the applicable processes/procedures during a phase are followed correctly or not.

Milestone Review:

Milestone Review is conducted on a periodic basis depending on the completion of a Particular phase or a defined timeframe. Examples**: reviews** may be a requirement review at the end of requirement phase, weekly status review at the end of week. or review percentage completion (say 5% project completion review).

Phase-End Review:

Phase-end review is conducted at the end of development such as requirements phase, design phase, coding, and testing, it be when there are distinct of development as defined in waterfall development model such as

requirements gathering, (high level and low level designs), coding, testing at various levels. deployment, etc. Phase-end reviews are also termed **'gate reviews'.**

Periodic Review:

Periodic review is done weekly, quarterly, monthly etc. Project Plan must define various periods when review of project related activities will be conducted.

Percent Completion:

Percent completion review is a combination of periodic and phase-end review where the project activities or product development activities are assessed on the basis of percent completion.

Process of In Process Review:

An organisation/project must have a definition of in-process It must have a list of stakeholders attending various reviews, and the work products under reviews. Commonly found steps of in-process reviews are given below

- Work product, and metrics undergoing are created and submitted to stakeholder.
- Inputs are received from stakeholders to initiate various actions.
- Risks identified by the project are reviewed and actions may be initiated as required.
- Any issue related to any stakeholder is noted and follow-up actions are initiated. Issues like software availability, hardware availability, training required etc.
- Review report is generated at the end, which helps in the next review.

B. Post Implementation Review:

A Post-Implementation Review (PIR) is conducted after completing a project. Its purpose is to evaluate whether project objectives were met, to determine how effectively the project was run, to learn lessons for the future, and to ensure that the organization gets the greatest possible benefit from the project.

Process of Post Implementation review:

- The key to a successful PIR is recognizing that the time spent on the project is just a small part of an ongoing timeline.
- For people and organizations that will be working on similar projects in the future, it makes sense to learn as many lessons as possible, so that mistakes are not repeated in future projects.

- And for organizations benefiting from the project, it makes sense to ensure that all desired benefits have been realized, and to understand what additional benefits can be achieved.

7.6 EXAMPLE OF ENTITIES INVOLVED IN VERIFICATION

Verification	Entities Involved	Definition
Requirement for Business / Functionality	Examine the business needs with the development team and the customer.	This is a crucial stage to ensure that the requirements have been acquired and/or accurately, as well as to determine whether or not they are realistic.
Design Review	Developer Team	The Developer team extensively evaluates the design once it is created to ensure that the functional requirements can be satisfied with the design presented.
Code Walkthrough	Individual Developer	After the code has been written, it is examined for any syntactic mistakes. This is a more informal task that is carried out by the individual developer on his or her own code.
Code Inspection	Developer Team	This is a more formal configuration. Subject matter experts and developers review the code to ensure that it meets the software's commercial and functional objectives.
Test Plan Review (internal to QA team)	Quality Analyst Team	The QA team reviews a test plan internally to ensure that it is correct and thorough.
Test Plan (External)	Project Manager, Business Analyst, and Developer	A formal review of the test plan document to ensure that the QA team's timeframe and other factors are in sync with the other teams and the overall project.
Test documentation	Quality Analyst Team	A peer review is when members of a team check each other's

review		work to ensure that the documentation is free of errors.
Test documentation final review	Business Analyst and Development Team	A review of the test documentation to ensure that the test cases cover all of the system's business circumstances and functional features.

7.7 REVIEWS IN TESTING LIFECYCLE

7.7.1. Test Readiness Review:

This is a very common activity that is performed by every QA team to determine whether they have everything they need to proceed into the test execution phase. Also, this is a recurring activity before each cycle of testing in projects that involve multiple cycles. In order to not run into issues after the testing phase begins and realize that we entered the execution phase prematurely, every QA project needs to conduct a review to determine that it has all the inputs necessary for successful testing. A checklist facilitates this activity perfectly. It lets you make a list of 'things-needed' ahead of time and to review each item sequentially.

Prerequisites Testing:

Software installation has some prerequisite testing such as operating system, database, reporting services as the case may exist. If prerequisites are not available installation may prompt or prerequisite will be installed during installation.

Updation Testing:

There must be a check whether similar operating systems already exist in the system or there is a need for a new version. Sometime it also prompts for repair or new installation.

Un-Installation Testing:

It is done when there is a need to check whether uninstallation is clean. When an application is uninstalled all the files installed must be removed from the disk.

7.7.2. Test Completion Review:

- Test Completion is the last stage of the software testing life cycle. It results in a report that a Test manager or a Test lead prepares that showcases the completed data from the test execution.
- The key activities carried out are
  - First is checking whether all reported defects reach a closure

- Second is creating a test summary report & communicating it to stakeholders.
- Next comes finalizing and archiving the test environment, the test data, the test infrastructure, and other test ware for later reuse.
- In addition to the above, analysing lessons learned from the finished test activities to determine changes needed for future iterations, releases, and projects happens.

7.8 COVERAGE IN VERIFICATION

7.8.1. Statement Coverage:

In this the testcase is executed in such a way that every statement of the code is executed at least once.

7.8.2. Branch/Decision Coverage:

Test coverage criteria requires enough test cases such that each condition in a decision takes on all possible outcomes at least once, and each point of entry to a program or subroutine is invoked at least once. That is, every branch (decision) taken each way, true and false. It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application.

7.8.3. Path Coverage:

In this the test case is executed in such a way that every path is executed at least once.

All possible control paths taken, including all loop paths taken zero, once, and multiple (ideally, maximum) items in path coverage technique, the test cases are prepared based on the logical complexity measure of a procedural design. In this type of testing every statement in the program is guaranteed to be executed at least one time. Flow Graph, Cyclomatic Complexity and Graph Metrics are used to arrive at the basis path.

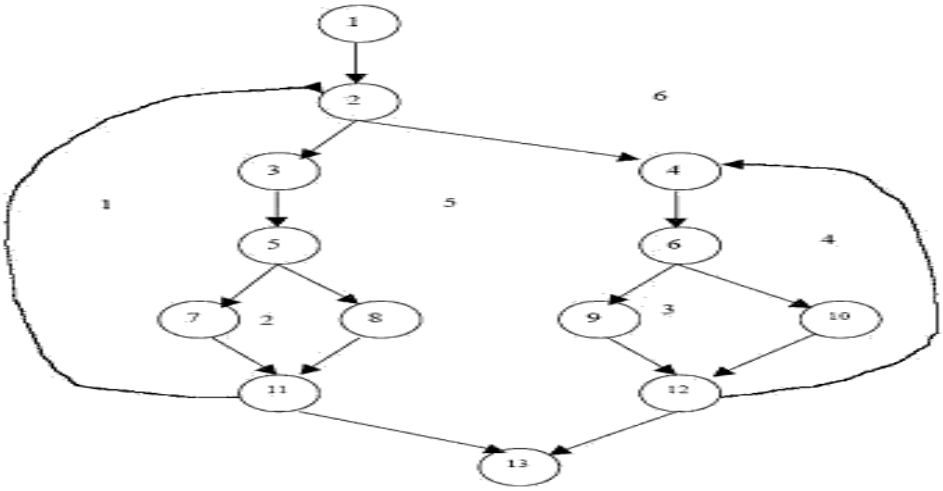
Cyclomatic Complexity:

Cyclomatic Complexity for a flow graph is computed in one of three ways:

1. The numbers of regions of the flow graph correspond to the Cyclomatic complexity.
2. Cyclomatic complexity, V(G), for a flow graph G is defined as **V(G) = E – N + 2**
3. where E is the number of flow graph edges and N is the number of flow graph nodes.
4. Cyclomatic complexity, V(G), for a graph flow G is also defined as **V(G) = P + 1**

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

Example: Consider the following flow graph



Region, R= 6

Number of Nodes = 13

Number of edges = 17

Number of Predicate Nodes = 5

Cyclomatic Complexity, V(C):

V (C) = R = 6;

Or

V(C) = Predicate Nodes + 1

=5+1 =6

Or

V(C)= E-N+2

= 17-13+2

7.8.4 DD Path Coverage:

- A **decision-to-decision path**, or **DD-path**, is a path of execution (usually through a flow graph representing a program) between two decisions.
- A DD-path is a set of nodes in a program graph such that one of the following holds
  - It consists of a single node with in-degree = 0 (initial node)
  - It consists of a single node with out-degree = 0 (terminal node)

- It consists of a single node with in-degree ≥ 2 or out-degree ≥ 2
- It consists of a single node with in-degree = 1 and out-degree = 1
- It is a maximal chain of length ≥ 1.

7.9 CONCERN OF VERIFICATION

There are few concerns associated with verification activities.

Use of right verification technique:

Every verification technique has advantages and disadvantages. For test cases, code file peer to peer review may be more advantageous from cost perspective while for requirement statement inspection technique can be used.

Integration of verification activities in SDLC:

As seen in V&V model every phase of SDLC has associated phases of verification and validation. Selection of verification technique must be such that it must find defects as early as possible. This can prevent stage contamination.

Resources and skill available for verification:

Most important resources and skills of verification are people and time. If reviewer is capable with sufficient knowledge and ability verification will be very effective.

7.10 VALIDATION

Validation is the process of checking whether the software product is up to the mark or in other words the product has high level requirements.

- It is the process of checking the validation of a product
- It also answers the question, **are we building the right product?**
- It is validation of actual and expected product.
- Validation is the **Dynamic Testing**.

7.10.1 Advantages of Validation:

- During verification if some defects are missed then during validation process it can be caught as failures.
- If during verification some specification is misunderstood and development has happened, then during the validation process while executing that functionality the difference between the actual result and expected result can be understood.

- Validation is done during testing like feature testing, integration testing, system testing, load testing, compatibility testing, stress testing, etc.
- Validation helps in building the right product as per the customer's requirement and helps in satisfying their needs.

7.10.2 Disadvantages of Validation:

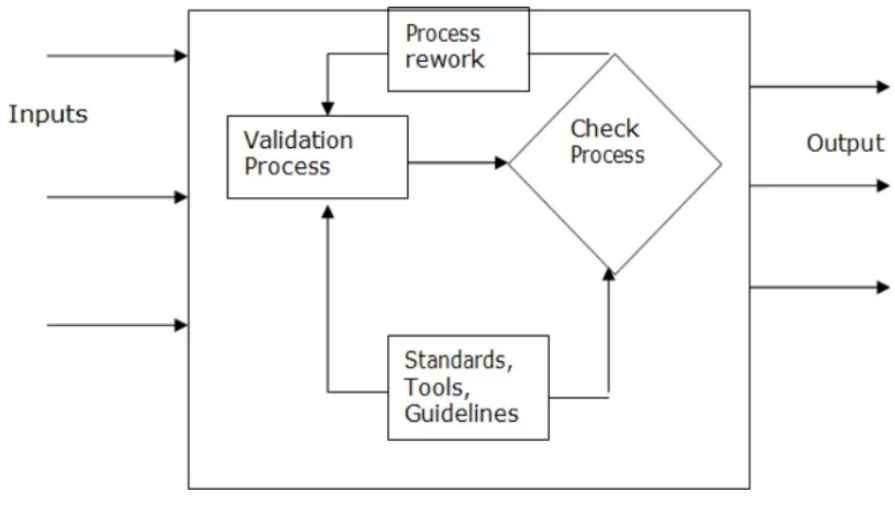
- No amount of testing can prove that software does not have defects. There can be many more defects not capture by the test cases,
- stub and driver are used during validation testing and they need additional efforts of development and testing before they can be used.

7.10.3 Prerequisites for validation

Prerequisites for validation may include the following.

- Training required for conduction the validation. Training may include domain knowledge and knowledge about testing and various test tools.
- Standard guidelines and tools to be used during validation.
- Validation do and check process definition.

7.11 VALIDATION WORKBENCH



Workbench is a place where validation activities are conducted on the work products and may be physical or virtual entities.

Inputs:

There must be some entry criteria definition when inputs are entering the work bench. This definition should match with the output criteria of the earlier work bench.

Outputs:

Similarly, there must be some exit criteria from the workbench which should match with input criteria for the next workbench. Outputs may include validated work products, defects, and test logs.

Validation:

Validation process must describe step-by-step activities to be conducted in a workbench. It must also describe the activities done while validating the work product under testing.

Check Process:

Check process must describe how the validation process has been checked. Test plan must define the objectives to be achieved during validation and check processes must verify that the objectives have been really achieved.

Standards tools and guidelines:

These may be termed 'the tools' available for validation. There may be testing guidelines or testing standards available.

7.12 LEVELS OF VALIDATION

1) Unit Testing:

- A Unit is a smallest testable portion of a system or application which can be compiled, loaded, and executed. This kind of testing helps to test each module separately.
- The aim is to test each part of the software by separating it. It checks that components are fulfilling functionalities or not. This kind of testing is performed by developers.

2) Integration testing

- Integration means combining. For Example, in this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.
- Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

3) System testing:

- System testing is performed on a complete, integrated system. It allows checking the system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

- System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional needs for the testing.

4) Acceptance testing

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

5) Interface Testing

- Interface Testing is defined as a software testing type which verifies whether the communication between two different software systems is done correctly.
- An interface is actually software that consists of sets of commands, messages, and other attributes that enable communication between a device and a user.

7.13 COVERAGE IN VALIDATION

Different instances of validation may offer different coverage depending on test plan and definition of coverage.

Following are some of the coverage.

1. Requirement Coverage:

- Requirements are defined in the requirement specification document. Traceability matrix start with requirement and goes forward up to test result.
- All requirements are not mandatory. They are put into different classes must, should be and could be. Most higher priority requirement is expressed as must, some lower priority is expressed as could be and should be.

2. Functionality Coverage:

Sometimes requirements are expressed in terms of functionality. Test cases representing higher priority must be tested to a larger extent than the functionality with the lower priority.

3. Feature Coverage:

- Features are groups of functionality doing same or similar things. It means covering at least one of the functionality which represent a feature provided even if there's multiple way of doing it.
- Feature with higher priority are indicated with must and lower one as should be or could be.

7.14 ACCEPTANCE TESTING

Acceptance testing is generally done by the users and/or customers to understand whether the software satisfies requirements or not, and to ascertain whether it is fit for use. Users\customers execute test cases to show if the acceptance criterion for the application has been met or not. There are three levels of acceptance testing.

7.14.1 Alpha Testing:

- Alpha testing represents the testing done by the customer in the development environment of the development team. The testing is done at a development Site with dummy data either created by or shared by the customer.
- In reality, alpha testing may be done by testers in front of the customer to show software is working. It can be used as a tool for training key users on the application.
- Any major drawback found can be cleared during alpha testing.

7.14.2 Beta Testing:

- Beta testing is used to assess the product by exposing it to the real end-users, usually called beta testers in their environment. Feedback is collected from the users and the defects are fixed.
- Also, this helps in enhancing the product to give a rich user experience.

7.14.3 Gamma Testing:

- Gamma Testing is the final stage of the testing process conducted before software release. It makes sure that the product is ready for market release according to all the specified requirements.
- Gamma testing focuses on software security and functionality. But it does not include any in-house QA activities.
- During gamma testing, the software does not undergo any modifications unless the detected bug is of a high priority and severity. Only a limited number of users perform gamma testing, and testers do not participate. The feedback obtained from the customer is used for new or enhanced products.

7.15 MANAGEMENT OF VERIFICATION AND VALIDATION (V & V)

Verification and validation techniques are complementary to each other, and not the replacement of each other.

The steps involved in verification/validation atv as follows.

7.15.1 Defining the Processes for Verification and Validation:

An organisation must define the processes applied for verification and validation activities during the development life cycle phase of the project. The processes involved may be as follows.

Software Quality Assurance Process:

These processes concentrate on developing the techniques/ procedures for development and testing of the project. They may be more generic, indicating an overall approach of handling verification and validation activities, or may be very specific for the project/customer.

Software Quality Control Process:

These processes concentrate on developing the approach for verification and validation activities during software development and testing. They can be more specific for the project under development. Quality control process forms a part of appraisal and failure cost.

Software Development Process:

These processes may concentrate over where zero defects introduce. They may cover software quality control process along with other SDLC processes.

Software life cycle definition:

It is essential to establish development and testing processes for the software.

7.15.2 Prepare Plans For Execution of Process:

When a project proposal is made, it must contain a definition of what is meant by a successful delivery, and must show how quality of deliverables will be achieved. The plans involved are as follows.

- a software development plan and schedule which include responsibilities and time schedule for verification/ validation activities must be defined at the start of the project.
- software quality plan and software test plan must define the use of different methodologies for ensuring quality of deliverables, verification and validation activities associated with development
- Software acceptance testing plan must be defined where acceptance criteria is finalised.

7.15.3 Initiate Implementation Plan:

- The plans made at the time of proposal/contract must be implemented during the development life cycle of a project. There must be formal records of requirements review, design review, code reviews, unit

testing, integration testing, interface testing, system testing, and acceptance testing.

- The results must be recorded and corrective/preventive actions must be initiated from the results of verification and validation.

7.15.4. Monitor Execution Plan:

The verification and validation activities must be monitored during development life cycle execution. Project manager and test manager must oversee that the activities defined in various plans are being executed and the results are being logged in the test log.

7.15.5 Analyse Problems Discovered During Execution:

Execution of verification and validation processes may bring out many problems in the product as well as Processes associated with development and testing. Root cause analysis of problems and planning for actions are essential parts of continuous improvement.

7.15.6 Report Progress:

The outcome of the validation and verification must be reported to the management, customer and development team to make them aware about project problems and progress.

7.15.7 Ensure Product Satisfies Requirement:

The ultimate aim of verification and validation activities undertaken during project execution is to achieve customer satisfaction.

7.16 SOFTWARE DEVELOPMENT VERIFICATION AND VALIDATION ACTIVITIES

Verification and validation activities are spread over the life cycle of software development. The activities include

Conceptualisation:

Conceptualìsation is the first phase of developing a product or project for a customer, Conceptualisation means converting the thoughts or concepts into reality. It can be through proof of concept or prototyping model. During proposal, the supplier may give of solution, and the customer may have to evaluate the feasibility of such an approach solution from product perspective. The supplier must understand what is expected by the customer, and whether it be provided or not by an organisation. Here verification and validation determine feasibility study of the project based on technical feasibility, economical feasibility and skill availability.

Requirement Analysis:

Requirement analysis phase starts from conceptualisation. Requirement understanding is done through communication with customer. It can be done- by various approaches joint application development and customer

survey. Requirement analysis verification and validation the feasibility of requirements and gives inputs to design approach. It can help in finding the gaps in proposals and requirements so that further clarifications can be achieved.

Design Requirements:

Design Requirements are implemented through design. Verification and validation of design understanding that design is complete in all respects and matches with requirements. Requirement ensures that all requirements are converted into design. It involves design through data flow diagram, prototyping.

Coding:

It involves code review and testing of the units, as part of verification and Validation, to make sure that requirements and design are correctly implemented. Units must be traceable to requirements through design.

Integration:

Integration validation and verification show that the individually tested units work correctly when brought together to form a module or sub module. It involves testing of modules/sub modules to ensure proper working with respect to requirements and designs. Integration with other hardware/ software is defined in architectural design. Interface testing must satisfy architectural design.

Testing:

Test artifacts such as test plan, test scenario, test bed, test cases, and test data must be subjected to verification and validation activities. Test plan must be complete, covering all aspects of testing expected by the customer. Test cases must cover the application completely. Test cases for acceptance testing must be validated by customer/user/business analyst.

Installation:

Application must be tested for installation, if installation is required by the customer, the documentation giving instructions for installation must be complete and sufficient to install the application, Verification and validation must ensure that there is adequate support and help available to common users for installation of an application.

Documentation:

There are many documents given along with a software product such as installation guide and user manual. They must be complete, detailed and informative, so that it can be referred to by a common user. The list of documents must be mentioned in contract or statement of work so that compliance can be checked.

7.17 LET US SUM UP

This chapter provides a clear exposition of verification and validation activities. It covers methods of verification such as reviews, walkthrough and inspection and various stages of verification such as requirement verification, design verification, and test artifacts verification. Advantages of different levels of reviews such as self review, peer review, walkthrough, and inspection have been dealt in detail. It also presents audits as an independent way. In-process reviews and post implementation reviews have also been evaluated. The Second half of the chapter focuses on validation techniques. The chapter concludes with a clear overview of acceptance testing.

7.18 EXERCISES

1. Discuss the advantages and disadvantages of verification
2. Describe different types of verification on the basis of parties involved in verification.
3. Explain self review
4. Describe the advantages and disadvantages of peer review.
5. Describe about walkthrough review
6. Explain formal review(Inspection).
7. Describe the process of inspection
8. Describe the auditing process.
9. What are the different audits planned during the development life cycle?
10. Explain various types of in-process review
11. Explain the concept of post-mortem review. it is essential for a learning organisation
12. Explain test readiness review.
13. What are the concerns of verification?
14. Discuss the advantages of validation
15. How coverage is measured in case of verification?
16. How coverage is measured in case of validation?
17. Discuss the different levels of validation.
18. Explain different levels of acceptance testing.

7.19 REFERENCES

- [Andriole86] Andriole, Stephen J., editor, *Software Validation, Verification, Testing, and Documentation*, Princeton, NJ: Petrocelli Books, 1986.
- http://tryqa.com/what-is-verification-in-software-testing-or-what-is-software-verification/
- https://www.tutorialspoint.com/software_testing_dictionary/audit.htm
- https://www.tutorialspoint.com/verification-and-validation-with-example
- https://qatestlab.com/resources/knowledge-center/alpha-beta-gamma/#:~:text=Gamma%20testing%20is%20the%20final,any%20in-house%20QA%20activities.
- M.G., LIMAYE, ed. (2009) *Software Testing: Principle, Technique and Tools*. Tata McGraw-Hill Publishing Ltd.

*****

8

V-TEST MODEL

Unit Structure

8.0 OBJECTIVES

After going through this chapter, you will be able to:

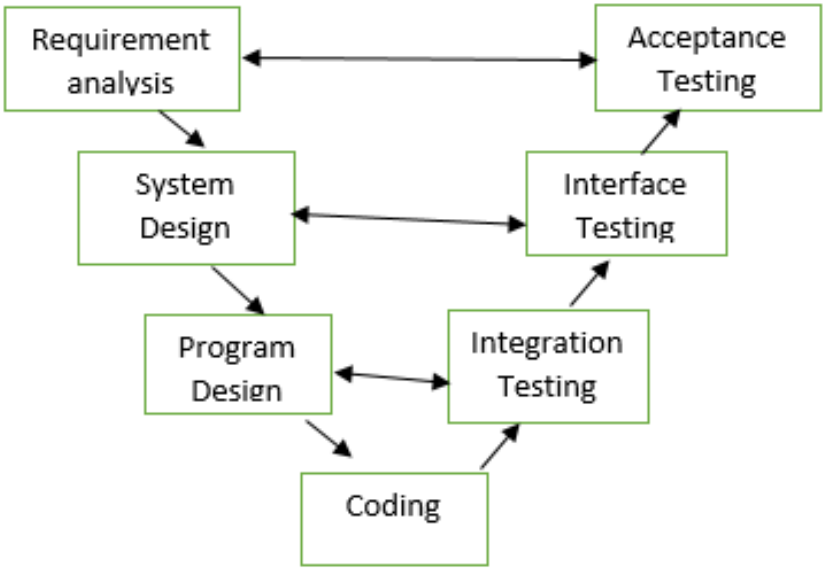
- V-Model (Validation Model)
- VV Model (Verification and Validation Model)
- Roles and Responsibilities of three critical entities in software development.

8.1 INTRODUCTION

Testing is a lifecycle activity. It starts when the proposal of software development is made to a prospect, and ends only when the application is finally delivered and accepted by the customer/end user. For product development, we may define each iteration of development as a separate project, and several projects may come together to make a complete product. For a customer, it starts from a problem statement or conceptualization of a new product, and ends with satisfactory product receipt, acceptance and usage. For every development activity, there is a testing activity associated with it, so that the phase achieves its milestone deliverable with minimum problem (theoretically, no problem). This is also termed 'certification approach of testing' or 'gate approach of testing'.

8.2 V MODEL FOR SOFTWARE

Validation model describes the validation activities associated with different phases of software development.



V-Model for Testing:

- Design phase is associated with interface testing which covers design specification testing as well as structural testing.
- Program-level designs are associated with integration testing.
- At code level, unit testing is done to validate individual units.

8.2.1 Structured Approach to Testing:

Testing activities for software development life cycle phases must be planned in advance, and conducted as per plan. Test policy and test strategy/test approach for performing verification and validation activities, responsibilities of stakeholders for supporting or doing these activities, inputs and outputs from each phase of the process, and milestone deliverables must be documented beforehand to avoid any problem in final deliverable to customer. Sometime it is also referred to V&V Plan.

8.2.2 Activities during Each Phase of Software Development Lifecycle:

Activities of verification and validation are planned during different phases of the software development lifecycle, from proposal level till product is finally accepted by the customer. The software development process plan must define the development and testing activities to be conducted in each phase of development also, the people responsible for conducting and supporting those activities as stakeholders. Plan of development must decide about 5 (What, Where, When, Why, and who) and H (How) with people, processes, tools, training, etc. The information

must be readily available, to the team doing these activities and the stakeholders, about their roles and responsibilities for those activities. Activities must be referred to during each phase of software development and testing.

8.2.3. Analyse Structures Produced During Development Phases for Adequacy and Testability:

Documents and work products produced during a life cycle phase must be analysed beforehand to understand their coverage, relationship with different entities, structure in overall development and traceability. An organization must have definition of processes, guideline and standard which can be used for making such documents and artifacts.

8.2.4 Generate Test Sets Based on Structures:

Functional test scenarios and test cases are generally developed based upon the functional requirements of the software. Functional requirements refer to the operational requirements of an application. **Structural test scenario** and test cases are developed from the structures defined in the design specifications. They must correspond to the structures of the work product produced during development. **Requirements/ designs** are verified and validated by preparing use case diagrams, data flow diagrams, and prototypes with the question 'what happens when' to identify the completeness of design and requirements. For validation testing scenarios, one must use techniques like boundary value analysis, equivalence partitioning, error guessing.

8.2.5 Additional Activities During Design And Coding:

Testing in low-level design and coding phases must confirm that first phase output of capturing the requirements and developing architecture matches with the inputs and outputs required by the low-level design phase. High-level design and low-level design must ensure that requirements are completely covered so that software developed covers all requirements. Verification and validation of design must ensure that the requirement verification and validation is proper and can be handled through the structures created for the purpose. Similarly, verification and validation of coding must ensure that all aspects of designs are covered by the code developed

8.2.6 Determine That Structures Are Consistent With Previously Generated Structures:

Software development is like a flow of events, starting from capturing the requirements till acceptance testing is completed successfully. The outputs of one phase must match with the input criteria of the next Phase. There must be consistency between various life-cycle phases. Consistency between various phases can improve the maintainability of an application.

8.2.7 Refine And Redefine Test Sets Generated Earlier:

Test artifacts such as test scenarios, test cases, and test data may be generated in each phase of software development life cycle from requirements, design, and coding, as the case may be. The test artifacts so generated must be reviewed and updated continuously as per the change in requirement, design etc.

8.3 TESTING DURING PROPOSAL STAGE

A proposal is created when the customer asks for information, quotation, and proposal. etc. At proposal stage, system description may not be very clear. People use different approaches such as the formal/informal, proof of concept. The Success of all these approach lies in successful defining the problem and proposed solution. Feasibility study is done by the customer as well as the supplier in search of a possible solution/approach to solve the problem faced by the customer. It may include technical feasibility, economic feasibility, implementation feasibility, organisational fit, process fit, and people fit.

8.4 TESTING DURING REQUIREMENT STAGE

Requirement gathering stage must cover all the requirements for the system. Characteristics of good requirements are mentioned below.

Cohesive:

The requirement defines a single aspect of the desired business process or system.

Complete:

The individual requirement is not missing necessary or relevant information. Additionally, the entire set of requirements should cover all relevant requirements.

Consistent:

The requirement does not contradict another requirement.

Modifiable:

Like requirements should be grouped together to allow similar requirements to be modified together in order to maintain consistency.

Correct: The requirement meets the actual business or system need. An incorrect requirement can still be implemented resulting in a business process or system that does not meet the business needs.

Observable:

The requirement defines an aspect of the system that can be noticed or observed by a user. This is often referred to as “Implementation Agnostic” as the requirement should not specify aspects of system architecture,

physical design or implementation decisions. These aspects of a system should be defined separately as constraints.

Feasible:

The requirement can be implemented within the constraints of the project including the agreed upon system architecture or other physical design or implementation decisions.

Unambiguous:

The requirement is written objectively such that there is only a single interpretation of the meaning of the requirement.

Verifiable:

It can be shown that the requirement has been met by the final solution via inspection, demonstration, test, or analysis.

8.5 TESTING DURING TEST-PLANNING PHASE

A Test Plan refers to a detailed document that catalogs the test strategy, objectives, schedule, estimations, deadlines, and the resources required for completing that particular project. Think of it as a blueprint for running the tests needed to ensure the software is working properly – controlled by test managers. Testing artifacts must be reviewed for their consistency and accuracy. Verification of testing artifacts may include the following.

Generate Test Plan to Support Development Activities:

Test plan must be consistent with the application development methodology, schedule, and deliverables. It must describe respective verification and validation activities to be conducted during each phase of the software development life cycle. It must contain methods used for defining test data, test cases, test scenario, and execution of testing activities. It must include how many defects would be expected to be found in a work product during testing.

Generate Test Cases Based on System Structure:

Functional test cases must be based on functional requirements, and structural test cases must be defined on the basis of design and non-functional requirements of the system. It must be used to define test data using different techniques available such as boundary value analysis, equivalence partitioning, error guessing, and state transition. Test cases and test data must be derived from test scenarios.

Analyse Requirement/Design Coverage:

It is very difficult for any test team to create test suite to cover 100% requirement and designs produced during software development life cycle. Coverage less than 100% indicates a risk. In case of any shortfall in coverage with respective objectives defined, the test team must analyse the

situation and perform risk-benefit analysis of lesser coverage with the help of customer and take corrective measures if required. Practically, it may not be possible, or it may not be required to cover all requirements.

8.6 TESTING DURING DESIGN PHASE

Design is the backbone of a software application. A successful design can convert the requirements into good application.

Consistency with respect to Requirements:

Designs must be consistent with defined. The requirement coverage of design must be analysed and measured. If there is no design for a given set of requirements, it will never get implemented. On the other hand, if there is a design component not referring to any requirements, it is considered as a defect.

Analyse Design for Errors:

Designs generated must be reviewed and tested for completeness and accuracy. A design is implemented by coding. Errors in the design will directly reflect the errors in coding and application so developed.

Analyse Error Handling:

Error handling of all kinds and possibilities must be covered in designing aspects. Organisation must have standards for error handling, error messaging and user interactions so that design is very clear about handling them during design.

Developers Verify Information Flow and Logical Structure:

Data flow diagrams or dummy execution of the system is done, and outputs derived are measured against expected outputs.

Testers Inspect Design in Detail:

Testers use design for defining structural test scenarios, test cases, and structural test data. Test scenario must be an end-to-end scenario considering data flow in the system.

8.6.1 Aspect to be tested:

Missing Test Cases:

Test cases not defined for a particular scenario (either or design) be caught in test-case review**.** Testers try to write the test scenario using the requirement statement without referring to design. This helps in the missing test cases.

Faulty Logic:

If the logic or algorithm described in design is not correct as per requirement statement, then it must be found by testing. Defects so found must be corrected by designers in design, and then by developers in code.

Module Interface Mismatch:

The data input/output from one module to another must be checked for consistency with design. Parameter passing is a major area of defects in software, where communication in two modules is affected. Interface test cases must test the scenario where one system is communicating with another system.

Data Structure Inconsistency:

Review of mismatch between data structures and definitions between different modules and system must be done for verification of designs

Erroneous Input/Output:

If the system needs to be protected from erroneous operations such as huge/invalid input/output, the design must describe how it will handle the situation.

Inconsistency with respect to Requirements and High-Level Design:

Design and development must be consistent with requirements and high-level design. As the system architecture is defined in high-level design which must reflect system requirements, any deviation can lead to extra or missing functionalities, or inappropriately implemented systems.

8.7 TESTING DURING CODING

Coding is the most crucial stage in software development where a product is actually built. Many organisations are completely dependent on self-review and peer review for verification of code. Also, unit testing is done by the developers. It is crucial but important to establish/institutionalize verification/validation of a code so that the product matches with requirement specifications.

8.7.1 Aspects to Be Checked:

Coding Standards/Guidelines Implementation:

Many organisations have coding Standards/ guidelines defined. When the code files are written, the developers must use these standards/guidelines. While reviewing code, the peer must make sure that Standards and guidelines are implemented correctly. It helps in optimization and better readability/maintainability of code in future.

Coding Optimization:

Coding standards must also talk about optimization of code. It talks about how nesting must be done, how declaration of variables and functions must be done.

Code Interpreting Design:

Coding must interpret designs correctly. Coding files and what they are supposed to implement must be defined in low-level design.

Unit Testing:

Unit testing must be done by the developers to ensure that written code is working as expected. Sometimes, unit testing is done by peer of an author (of a code) to maintain independence of testing with respect to development. Unit test case logs must be prepared and available for peer review, SQA review as well as customer audits.

8.8 VV MODEL

Quality checking involves verification as well as validation activities. 'VV model' considers all the activities related to verification as well as validation. It is also termed as 'Verification and Validation Model' or 'Quality model'.'VV model' talks about verification and validation activities associated with software development during the entire life cycle.

Requirements:

As requirements are obtained from customer like customer survey, prototyping. The intention would be to find if there is any gap existing between user requirements and requirement definition.

1. Requirement Verification:

a. Verification tests check to ensure the program is built according to the stated requirements. The verification process includes activities such as reviewing the code and doing walkthroughs and inspections.

b. Missing requirements or invalid requirements can be discovered during this phase, which can minimize the risk of rework and the cost associated with overruns. It’s far more effective to fix a small bug upfront than in the future when hundreds of lines of code must be identified and corrected

2. Requirement Validation:

a. It ensures that the requirements have achieved the business objectives, meet the needs of any relevant stakeholders and are clearly understood by developers. Validation is a critical step to finding missing requirements and ensuring that requirements have a variety of important characteristics.

b. Software validation addresses the following:

I. Correctly outlines the end user’s needs.

II. Has only one exact meaning.

III. Can be modified as necessary.

IV. Documents the attributes that customers truly need.

V. Easily linked to system requirements, such as designs, codes and tests.

Design:

Design may include high-level design or architectural design, and low-level design or detail design. Designs are created by architects (high-level designs)/designers (low-level designs) as the case may be.

a. Design Verification:

Verification of design may be a walkthrough of a design document by design experts, team members and stakeholders of the project. Project team along with the architect/designer may walkthrough the design to find the completeness and give comments, if any. Traceability of design with requirement must be established in the requirement traceability matrix.

b. Design Validation:

Validation of design can happen at two stages.

i. The first stage is at the time of creation of data flow diagram. If flow of data is complete, design is complete. Any problem in flow of data indicates gap in design. This is term as flow anomaly.

ii. The second stage of validation is at integration or interface testing. Integration testing is an activity to bring the units together and test them as a module. Interface testing is an activity of testing connectivity and communication of application with the outside world. For eg: the module connection with outside world like database, browser, and operating system.

Coding:

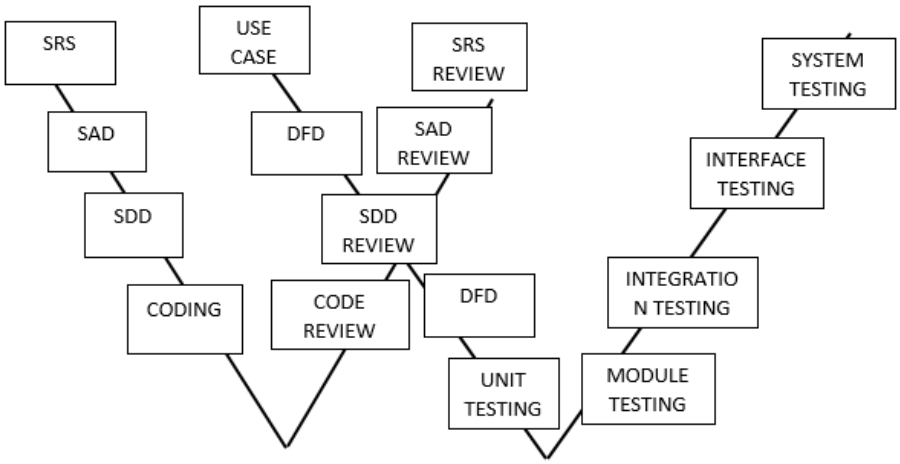
Coding is an activity of writing individual units as defined in low level design.

a. Code Verification:

As coding is done, it undergoes a code review (generally peer review). Peer helps in identification of errors with respect to coding standards, indenting standards, commenting standards and variable declaration issues. The Checklist approach is used in code review.

b. Code Validation:

Validation of coding happens through unit testing where individual units are separately The developer may have to write special programs (such as stubs and drivers) so that units can be tested. The executable is created by combining stubs, drivers and the units under testing.



VV MODEL

8.9 CRITICAL ROLES AND RESPONSIBILITIES

In software verification and validation, three critical roles can be identified.

Development:

Development team may have various roles under them. They may be performing various activities as per the role and responsibilities at various stages. Some of the activities related to development group may be as follows

- Project Planning activities include requirement elicitation, project planning scheduling. Project planning is the foundation of success.
- Resources may include identification and organization of adequate number of people, machine, hardware, software and tools required by the project. It may also include assessment of skills required by plans.
- Interacting with customers and stakeholders as per project requirement.
- Defining policies and procedures for creating development work, verification and validation activities so that quality is built properly, delivering it to test team/customer as the case may be.

Testing:

Testing team may include test manager, test leads, and testers as per scope of testing, size of project, and type of customer. Generally, it is expected that a test team would have independence of working and they do not have to report to the development team.

Roles and responsibilities of test team may include the following:

- Test planning including test strategy definition, and test case writing. Test planning may include estimation of efforts and resources required for testing.
- Resourcing may include identification and organisation of adequate numbers of people, machines, hardware, software, and tools as required by the project. It may also involve an assessment of skills required by the project, skills already available with them and any training needs.
- Defining policies and procedures for creating and executing tests as per test strategy and test plan. Testers may have to take part in verification and validation activities related to test artifacts.
- Doing acceptance testing related activities such as training and mentoring to users from customer side before/during acceptance testing activities.

Customer:

Customer may be the final user group, or people who are actually sponsoring the project. Customers can be internal to an organisation or external to the organisation. Roles and responsibilities of a customer may include the following.

- Specifying requirements and signing off requirement statements and designs as per contract. It may also include solving any queries or issues raised by the development/test team.
- Participating in acceptance testing as per roles and responsibilities defined in the acceptance test plan. A customer may be responsible for alpha, beta and gamma testing, as the case may be.

8.10 LET US SUM UP

This chapter provides a clear exposition of V-MODEL and VV Model (Verification and Validation). It covers activities of verification and validation along with its role and responsibilities. The chapter concludes with a clear overview of roles and responsibilities which is important for the good product.

8.11 EXERCISES

1. What are the characteristics of good requirements?
2. Describe V & V activities during the proposal.
3. Describe V & V activities during requirement generation.
4. Describe V & V activities for test artifacts.

5. Describe V & V activities during designs.
6. Describe V & V activities during coding.
7. Explain VV Model
8. What are the role and responsibilities in software verification and validation

8.12 REFERENCES

- [Andriole86] Andriole, Stephen J., editor, Software Validation, Verification, Testing, and Documentation, Princeton, NJ: Petrocelli Books, 1986.
- https://www.softwaretestinghelp.com/what-is-stlc-v-model/
- M.G., LIMAYE, ed. (2009) Software Testing: Principle, Technique and Tools. Tata McGraw-Hill Publishing Ltd.

*****

UNIT - V

9

LEVELS OF TESTING

Unit Structure

9.0 OBJECTIVES

After going through this chapter, you will be able to:

- describe various verification and validation activities associated with various stages of SDLC starting from proposal.
- discuss various approaches of integration testing.

9.1 INTRODUCTION

Testing is a life-cycle activity which begins with a proposal for software/system application and ends when product is finally delivered to customer. Different agencies/stakeholders are involved in conducting specialised testing required for the specific application. Definition of these stakeholders/agencies depends on the type of testing involved and level of

testing to be done in various stages of software development life cycle. Table 9.1 shows in brief the level of testing performed by different agencies. It is an indicative list which may differ from customer to customer, product to product and organisation to organisation.

Table 9.1 Different agencies involved at different levels of testing

Testing	Agencies involved
Proposal review	Customer, Business Analyst, System Analyst, Project Manager
Proposal testing	Customer, Business Analyst, System Analyst, Project Manager
Requirement review	Customer, Business Analyst, System Analyst, Project Manager, Project Leader, Test Leader
Requirements testing	Customer, Business Analyst, System Analyst, Project Manager, Project Leader, Test Leader
Design review	Customer, Business Analyst, System Analyst, Architect, Project Manager, Project Leader, Test Leader, Developer
Design testing	Customer, Business Analyst, System Analyst, Architect, Project Manager, Project Leader, Test Leader, Developer
Code review	Project Leader, Project Team, Customer
Test artifact review	Project Leader, Project Team, Customer, Tester, Test Leader
Unit testing	Project Leader, Project Team, Customer
Module testing	Project Leader, Project Team, Tester, Customer
Integration testing	Project Leader, Project Team, Tester, Customer
System testing	Project Manager, Test Leader, Tester, Customer
Acceptance testing	Project Manager, Tester, User/Customer

9.2 PROPOSAL TESTING

A proposal is made to the customer on the basis of Request for Proposal (RFP) or Request for Information (RFI) or Request for Quotation (RFQ) by the customer. Before making any proposal, the supplier must understand the purpose of such request, and devise the proposed solution accordingly. One must understand customer problem and the possible solution. Any proposal prepared in response to such request is reviewed by an organisation before sending it to the customer. It is reviewed by different panels or groups in the organisation such as technical group and commercial group. The intention is to ensure that the organisation must be able to stand by its words, if the proposal gets converted into a contract. The organisation must assess the impact of proposal on the organisation as well as on the prospect/customer.

Technical Review:

Technical review mainly involves technical feasibility of the kind of system or application proposed, the availability and requirement of skill sets, the hardware/software and other requirements of system. The proposal is also reviewed on the basis of time required for developing such system, and efforts required to make the proposal successful. The technical proposal is a description of overall approach, and not the final accepted approach solution/method and may undergo many iterations of changes as per discussions with customer. Requirement and estimations at the stage of proposal are not very specific but rather, they are indicative. It works on rough-cut methods, and estimations are ball-park figures that prospect may expect.

Commercial Review:

A proposal undergoes financial feasibility and other types of feasibilities involved with respect to the business. Commercial review may stress on the gross margins of the project and fund flow in terms of money going out and coming in the development organisation. Generally, total payment is split into installations depending upon completion of some phases of development activity. As different phases are completed, money is realised at those instances.

Several iterations of proposals and scope changes may happen before all the parties involved in Request for Quotation (RFQ) and proposal agree on some terms and conditions for doing a project.

Validation of Proposal:

A proposal sometimes involves development of prototypes or proof of concept to explain the proposed approach to customer problem. One must define the approach of handling customer problem in the model or prototype.

In case of product organisation, the product development group along with marketing and sales functions decides about the new release of a product.

9.3 REQUIREMENT TESTING

Requirement creation involves gathering customer requirements and arranging them in a form to verify and validate them. The requirements may be categorised into different types such as technical, economical, legal, operational, and system requirements. Similarly, requirements may be specific, generic present, future, expressed, implied, etc. The customer may be unaware of many of these requirements. The business analyst and system analyst must study the customer's line of business, problem and solution that the customer is looking for before arriving at these requirements. Assumed, implied or intended requirement is a gray area and may be a reason for many defects in the final product. Requirement testing makes sure that requirements defined in requirement specifications meet the following criteria:

Clarity:

All requirements must be very clear in their meaning and what is expected from them. Requirements must state very clearly what the expected result of each transaction is, and who are the actors taking part in the transactions.

Complete:

Requirement statement must be complete and must talk about all business aspects of the application under development. Any assumption must be documented and approved by the customer.

Measurable:

Requirements must be measurable in numeric terms as far as possible. Such definition helps to understand whether the requirement has been met or not while testing the application. Qualitative terms like 'good', 'sufficient', and 'high' must be avoided as different people may attach different meanings to these words.

Testable:

The requirement must help in creating use cases/scenario which can be used for testing the application. Any requirement which cannot be tested must be drilled down further, to understand what is to be achieved by implementing these requirements.

Not Conflicting:

The requirements must not conflict with each other. There is a possibility of trade-off between quality factors which needs to be agreed by the customer. Conflicting requirements indicate problem in requirement collection process.

Identifiable:

The requirements must be distinctively identifiable. They must have numbers or some other way which can help in creating requirement traceability matrix. Indexing requirements is very important.

Validation of Requirements:

In requirement testing, clear and complete use cases are developed using requirement statement. Any assumption made indicates lacunae in requirement statement and it should be updated by verifying the assumptions made with the customer.

9.4 DESIGN TESTING

Once the requirement statement satisfies all characteristics defined above, the system architect starts with system architecture design. The designs made by system architects must be traceable to requirements. The design must also possess the characteristics given below:

Clarity:

A design must define all functions, components, tables, stored procedures, and reusable components very clearly. It must define any interdependence between different components and inputs/outputs from each module. It must cover the information to and from the application to other systems.

Complete:

A design must be complete in all respect. It must define the parameters to be passed/received, formats of data handled, etc. Once the design is finalised, programmers must do their work of implementing designs mechanically and system must be working properly.

Traceable:

A design must be traceable to requirements. The second column of requirement traceability matrix is a design column. The project manager must check if there is any requirement which does not have corresponding design or vice versa. It indicates a defect if traceability cannot be established completely.

Implementable:

A design must be made in such a way that it can be implemented easily with selected technology and system. It must guide the developers in coding and compiling the code. The design must include interface requirements where different components are communicating with each other and with other systems.

Testable:

A good design must help testers in creating structural test cases.

Validation of Design:

Design testing includes creation of data flow diagrams, activity diagrams, information flow diagram, and state transition diagram to show that information can flow through the system completely. Flow of data and information in the system must complete the loop. Another way of testing design is creating prototypes from design.

9.5 CODE REVIEW

Code reviews include reviewing code files, database schema, classes, object definitions, procedures, and methods. Code review is applied to ensure that the design is implemented correctly by the developers, and guidelines and standards available for the coding purpose are followed correctly. Code must have following characteristics:

Clarity:

Code must be written correctly as per coding standards and syntax, requirements for the given platform. It must follow the guidelines and

standards defined by the organisation/project/customer, as the case may be. It must declare variables, functions, and loops very clearly with proper comments. Code commenting must include which design part has been implemented, author, date, revision number, etc. so that it can be traced to requirements and design.

Complete:

Code, class, procedure, and method must be complete in all respect. One class doing multiple things, or multiple objects created for same purpose indicates a problem in design.

Traceable:

Code must be traceable with design components. Code files having no traceability to design can be considered as redundant code which will never get executed even if design is correct.

Maintainable:

Code must be maintainable. Any developer with basic knowledge and training about coding must be able to handle the code in future while maintaining or bug fixing it.

9.6 UNIT TESTING

Unit is the smallest part of a software system which is testable. It may include code files, classes and methods which can be tested individually for correctness. Unit testing is a validation technique using black box methodology. Black box testing mainly concentrates on system requirements.

- Individual components and units are tested to ensure that they work correctly as an individual as defined in design.
- Unit testing requires throwaway drivers and stubs as individual files may not be testable or executable without them.
- Unit testing may be performed in debugger mode to find how the variable values are changed during the execution. But, it may not be termed ‘black box testing’ in such case as code is seen in debugging.
- Gray box testing is also considered as ‘unit testing technique’ sometimes as it examines the code in detail along with its functioning. Gray box testing may need some tools which can check the code and functionality at the same time.
- Unit test cases must be derived from use cases/design component used at lowest levels of design.

9.6.1 Difference between Debugging and Testing:

Many developers consider unit testing and debugging as the same thing. In reality there is no connection between the two. Difference between them can be illustrated as shown in Table 9.2

Table 9.2 Difference between debugging and unit testing

Debugging	Unit testing
It involves code checking to locate the causes of defect.	It checks the defect and not the causes of the defect.
Code may be updated during debugging.	It does not involve any code correction.
Test cases are not defined.	Test cases are defined based on requirements and design.
Generally covers positive cases to see whether unit works correctly or not.	Covers positive as well as negative cases.

9.7 MODULE TESTING

Many units come together to form a module. The module may work on its own or may need stubs/drivers for its execution. If the module can work independently, it is tested by tester. If it needs stubs and drivers, developers must create the same and perform testing. Module testing mainly concentrates on the system structure.

- Module testing is done on related unit-tested components to find whether individually tested units can work together as a module or not.
- Module test cases must be traceable to requirements/design.

9.8 INTEGRATION TESTING

Integration testing involves integration of units to make a module/integration of modules to make a system/integration of system with environmental variables if required to create a real-life application.

Integration testing may start at module level, where different units and components come together to form a module, and go up to system level. If module is self-executable, it may be tested by testers. If it needs stubs and drivers, it is tested by developers.

Though integration testing also tests the functionality of software under review, the main stress of integration testing is on the interfaces between different modules/systems. Integration testing mainly focuses on input/output protocols, and parameters passing between different units, modules and/or system. Focus of integration is mainly on low-level design, architecture, and construction of software. Integration testing is

considered as 'structural testing'. Figure 9.1 shows a system schematically.

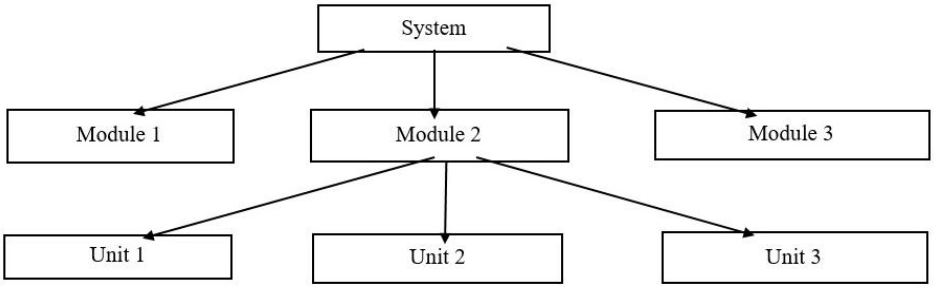


Fig. 9.1 Integration testing view

There are various approaches of integration testing depending upon how the system is integrated. Different approaches have different benefits and limitations.

9.8.1 Bottom-Up Testing:

Bottom-up testing approach focuses on testing the bottom part/individual units and modules, and then goes upward by integrating tested and working units and modules for system testing and intersystem testing. Figure 9.2 shows bottom-up integration and testing.

- Units at the lowest level are tested using stubs/drivers. Stubs and drivers are designed for a special purpose. They must be tested before using them for unit testing.
- Once the units are tested and found to be working, they are combined to form modules. Modules may need only drivers, as the low-level units which are already tested and found to be working may act as stubs.
- If required, drivers and stubs are designed for testing as one goes upward. Developers may write the stubs and drivers as the input/output parameters must be known while designing.
- Bottom-up approach is also termed 'classical approach' as it may indicate a normal way of doing things. Theoretically, it is an excellent approach but practically, it is very difficult to implement.
- Each component which is lowest in the hierarchy is tested first and then next level is taken. This gives very robust and defect-free software. But, it is a time consuming approach.

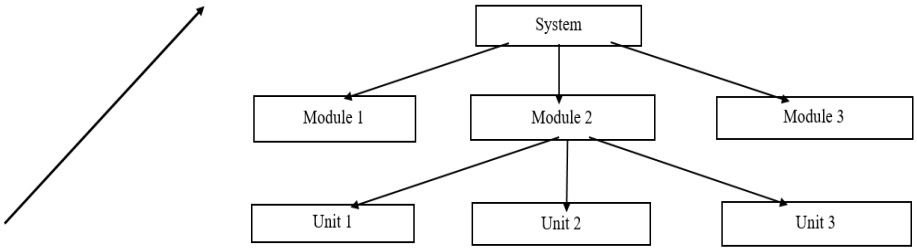


Fig. 9.2 Bottom-up integration approach

Bottom-up approach is suitable for the following:

- Object-oriented design where objects are designed and tested individually before using them in a system. When the system calls the same object, we know that they are working correctly (as they are already tested and found to be working in unit testing).
- Low-level components are general-purpose utility routines which are used across the application; bottom-up testing is the recommended approach. This approach is used extensively in defining libraries.

Stubs/Drivers:

- Stubs/drivers are special-purpose arrangements, generally code, required to test the units individually which can act as an input to the unit/module and can take output from the unit/module.

Stub:

Stub is a piece of code emulating a called function. In absence of a called function, stub may take care of that part for testing purpose.

Driver:

Driver is a piece of code emulating a calling function. In absence of actual function calling the piece of code under testing, driver tries to work as a calling function.

- Stubs are mainly created for integration testing like top-down approach. Drivers are mainly created for integration testing like bottom-up approach.
- Stubs/drivers must be very simple to develop and use. They must not introduce any defect in the application. They should be eliminated before delivering code to customer.
- Creating reusable stub/driver improves productivity, but it is a big challenge.

Advantages of Bottom-Up Approach:

- Robust system, as each unit is tested individually.

Disadvantages of Bottom-Up Approach:

- Slower process.
- Most important components at the top-level are tested at the end, might result in product delivery pressure.
- Wastage of effort in designing stubs and drivers.

9.8.2 top-Down Testing:

Top-down testing begins with testing of top-level components of the application, like user interfaces and then proceeds downward till it reaches the final component of the system. Here, integration goes downward as the tested components at the top are combined one by one in the process. Top-down approach needs design and implementation of stubs. Agile models like prototyping use this testing approach. Figure 9.3 shows top-down integration approach.

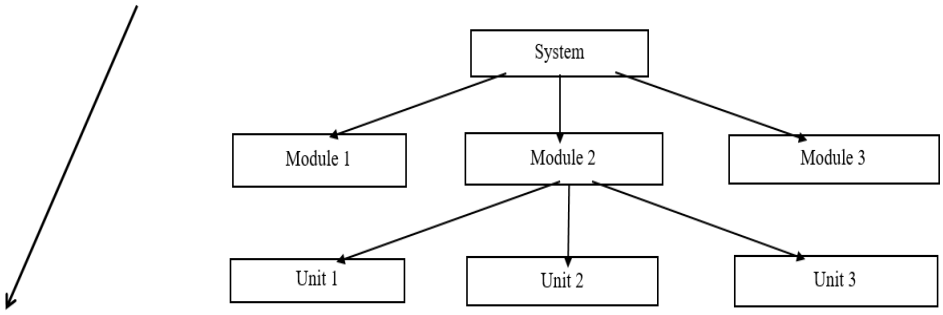


Fig. 9.3 Top-down integration approach

Advantages of Top-Down Approach:

- This approach begins from top layer thereby determining feasibility of application at an early stage.
- Major flaws can be detected by taking user inputs.
- Generally, this approach does not need drivers as top-layers can work as drivers for bottom layers.

Disadvantages of Top-Down Approach:

- Sometimes individual units/module flaws remain hidden as they are rarely tested alone.
- This approach can create a false belief among customer that software is ready as prototype of software is delivered to customer even before design phase.
- Wastage of effort in designing large number of stubs.

9.8.3 Modified Top-Down Approach:

Modified top-down approach combines the advantages of bottom-up and top-down approaches and tries to remove the disadvantages of both. This approach begins testing from both the ends to meet somewhere in between. One part of testing starts from top to bottom as we are integrating units downward, and the second part from bottom to top for selected components declared as 'critical units'. The main challenge is to determine individual unit/module for bottom-up testing. Configuration management is very important for such an approach. This approach makes the system testing twice. This may require more resources and more time than top-down approach but lesser than bottom-up approach. This approach is more practical as all components are not equally important. Figure 9.4 shows a modified top-down approach.

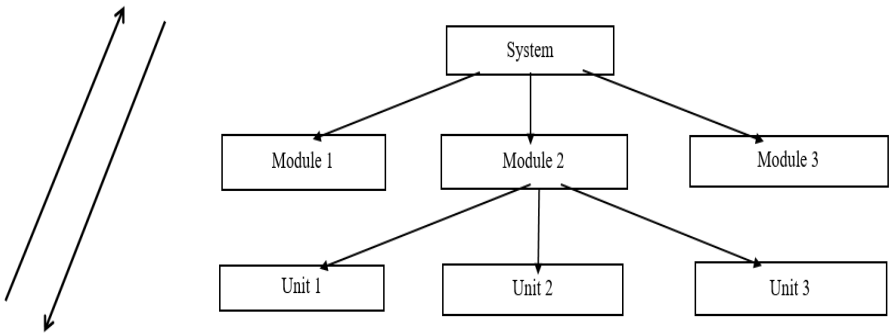


Fig. 9.4 Modified Top-down integration approach

Advantages of Modified Top-Down Approach:

- Important units are tested individually, and then combined to form the modules, and finally modules are tested to make the system. This is done during unit testing followed by integration testing.
- This approach is time saving as all components are not tested individually. This approach believes not all components are critical for a system.
- The systems tested using this approach are more efficient as compared to top-down and bottom-up approach.

Disadvantages of Modified Top-Down Approach:

- Stubs and drivers are required.
- Identification of critical units is challenging.

9.9 BIG-BANG TESTING

The system is tested completely after development in big-bang approach. Individual units/modules are not tested.

Advantages of Big-Bang Approach:

- Since testing is done at the end phase, therefore time required for writing test cases and defining test data at unit level, integration level, etc. may be saved.
- This approach can be used as a validation of development process if an organisation has optimized processes. If all testing phases are successfully completed, then system testing may act as certification testing before final product delivery.
- No stub/driver is required in this approach. It is cost efficient approach.
- Fast approach.

Disadvantages of Big-Bang Approach:

- Defects found during testing are difficult to trace and debug.
- This approach executes the test cases without any understanding of how the system is build.
- Interface faults may not be distinguishable from other faults.
- Based on just few test cases, testers certify the system.

9.10 SANDWICH TESTING

Sandwich testing conducts testing into two parts, and follows both parts starting from both ends i.e. top-down approach and bottom-up approach either simultaneously or one after another. It combines the advantages of both these approaches. Figure 9.5 shows sandwich testing approach.

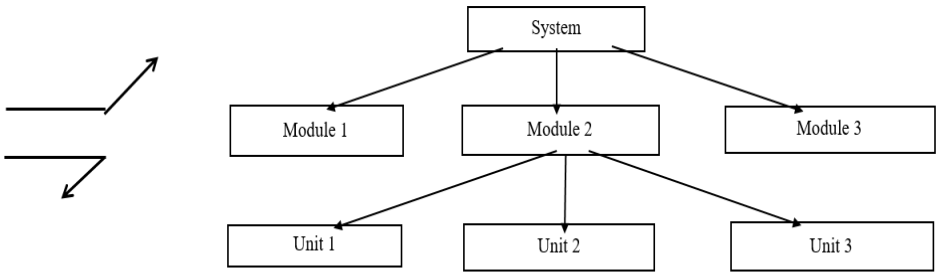


Fig. 9.5 Sandwich testing approach

Process of Sandwich Testing:

- Bottom-up testing starts from middle layer and moves upward. Top-down testing starts from middle layer and goes downward.
- Big-bang approach is followed for the middle layer. From this layer, bottom-up approach goes upwards and top-down approach goes downward.

Advantages of Sandwich Testing:

- Suitable for large projects following spiral model.
- Both top-down and bottom-up testing begin simultaneously.

Disadvantages of Sandwich Testing:

- Higher testing costs as it requires more resources and big teams for performing top-down and bottom-up testing simultaneously.
- Not suitable for smaller projects.
- Skilled testers with varying skill sets are required as different domains representing different functional areas have to be handled.

9.11 CRITICAL PATH FIRST

In critical path first, critical path is determined which must be covered first. Critical path represents the main function of a system. Development team focuses on design, implementation and testing of critical path of a system first. This is also called 'skeleton development and testing'. In order to define critical path correctly, it is important to understand the system criticality from user's perspective or from business perspective and then decide on priority of testing. This approach is generally applicable when entire system testing is impossible and systems are large enough.

9.12 SUBSYSTEM TESTING

Subsystem testing involves collection of units, sub modules, and modules which have been integrated to form subsystems. It mainly concentrates on detection of integration and interface errors

9.13 SYSTEM TESTING

System testing is the final stage of testing just before system is delivered to customer. It validates the fulfilment of functional/non-functional requirements as specified in software requirement specification. It involves following stages:

Functional Testing:

System software is expected to perform certain functions as mentioned in requirement specification. This testing checks working of these intended functionalities and corrects defects, if any.

User Interface Testing:

After functional testing, user interface is tested if the system has any user interface. This testing may involve colors, navigations, spellings, and fonts.

Once functional and user interface testing is completed, system undergoes other types of testing like security, load, and compatibility.

9.14 TESTING STAGES

First, unit testing is performed to identify and eliminate any defects at unit level. Second, integration testing or module testing is done on integrated units. After module testing, subsystem testing is executed. Then system testing is performed after the system completes all levels of integration. In some cases, interface testing is done instead of system testing to fix any issues with inter system communication. Lastly, acceptance testing is done to check whether acceptance criteria is fulfilled or not. Testing stages are shown in Fig. 9.6.

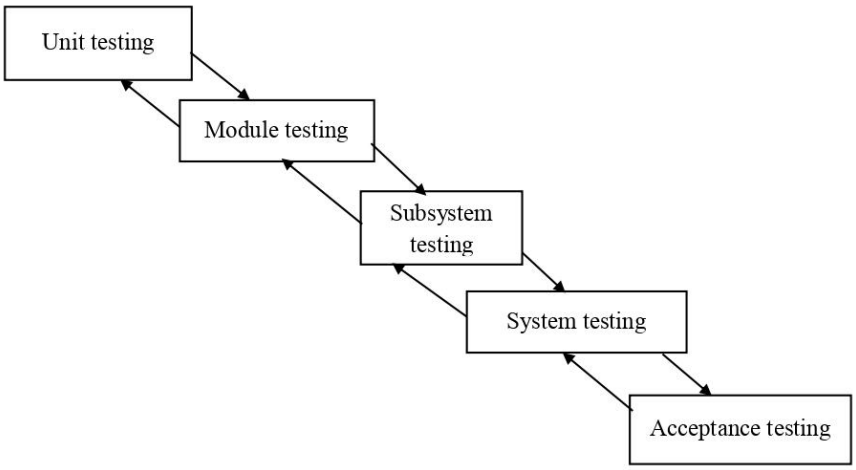


Fig. 9.6 Software testing stages

9.15 SUMMARY

This chapter describes verification and validation activities related to various stages of software development phases. It discusses merits and demerits of various integration testing approaches.

9.16 EXCERCISES

1. Describe proposal review process.
2. Describe requirement verification and validation process.
3. Describe design verification and validation process.
4. Describe code review and unit testing process.
5. Describe difference between debugging and unit testing.
6. Differentiate between integration testing and interface testing.
7. Describe bottom-up approach for integration.
8. Describe top-down approach for integration.
9. Describe modified top-down approach for integration.

*****

10

SPECIAL TESTS

Unit Structure

10.0 OBJECTIVES

After going through this chapter, you will be able to:

- Explain special techniques other than system testing which may be required as per customer or application specification.
- Understand requirement statement clearly. Know and comprehend the testing scope, application type, and customer's line of business before devising any test.
- Understand the support documentation and tools required for doing testing.
- Understand the sequence of activities to be done while conducting different tests.

10.1 INTRODUCTION

Testing includes not just functionality testing. Depending on application and requirement specifications, some specialized tests may be conducted. Special testing may need some special testing skills, tools and techniques.

Some systems are intended for specific purpose and they need to fulfill special characteristics mentioned in requirement statement. Testing of such systems is included in this type of testing. Examples include eBusiness system, security system, antivirus applications, administrative system, operating systems and databases. Such systems are created for specific users and may or may not cater to all in general.

10.2 GRAPHICAL USER INTERFACE TESTING

User interacts with the system using user interface. Graphical user interface is an important part of the application affecting usability. Graphical user interface testing is also known as 'GUI' testing or 'UI' testing. Application system testing begins with functionality testing which is followed by GUI testing. Graphical user interface includes the following:

- Colors used for background, control and font should be chosen carefully as they directly impact users.
- Words, fonts, and alignments used on screen also have a huge impact on user readability and interaction with system. Font size and alignments should be set properly. Spelling mistakes should not be there.
- Scrolling pages up and down as well as navigation for different hyperlinks and pages must be avoided as much as possible as it increases frustration in users and reduces usability. Page layout should be maintained.
- Error and alert messages must be meaningful and easily understandable to users.
- Reports and print outs should take readability, font, screen size, paper size on printer into consideration.
- Screen layout in terms of number of instructions to users, number of controls and number of pages are defined in low-level design. Too many pages and many controls on single page reduce application usability.
- Appropriate control selection improves operability of application like placing drop down control instead of free text. Logical placement of controls and tab sequence maintenance improves usability and screen appearance.
- Huge graphics impact the loading time of page. Pages must be kept simple for easy loading/unloading in web application.

Advantages of GUI Testing:

- GUI impacts user's emotion and psychological impact of using an application. It is essential to have a good GUI to improve look and feel of the application.

- GUI represents the presentation of an application. Prototyping is used extensively for verifying GUI requirements.
- Placing 'Help' option ensures the users quick access to help in case of any problem resulting in enhanced user experience and improved application reliability.
- Consistency of screen layouts, appropriate tab sequence, and designs improves application usability.

Disadvantages of GUI Testing:

- Due to huge controls on a page and large number of pages, spelling mistakes might go unnoticed by tester.
- Special applications targeted for specific end users, like blind people or kids below 5 years of age may require software testers to be trained to behave like target users.
- Functionality testing is prioritized over GUI testing resulting in negligence of GUI defects.
- GUI testing is not given much importance and often assigned to junior testers.

10.3 COMPATIBILITY TESTING

Application is generally made up of different components which work together to achieve a goal. In software systems, these components may be printer, hardware, operating system, databases etc. Market size is impacted by the product and its operability with components. Product vendors always aim to maximize the working of product on every possible scenario of different components. Compatibility testing refers to testing the software on multiple configurations to check the behavior of different system components and their combinations. Different types of compatibility testing are performed.

10.3.1 Multiplatform Testing:

Software testing on different platforms is done in Multiplatform testing. Performance of software must not be impacted as the platform is changed. Platform may be freeware, open source or licensed. Customers prefer to use applications running on their existing platform. Product vendors will always prefer to have maximum possible platform coverage so that they can have customers from across the globe.

- In multiplatform testing, validation of software execution across different platforms is done, also called parallel testing. Verification is done by comparing the results of processing on different platforms. The aim is to ensure that software achieves its functionalities on every platform.

- The platform on which software development is done is referred as 'base platform' and it is expected to behave in similar manner on range of platforms. This range of platforms must be clearly defined in requirement statement and they should be already existing platforms so that testing can be performed.
- When software is working on different platforms, some performance variations are allowed and this range of performance variations allowable must be mentioned in requirement statement.

Major Concerns in Multiplatform Testing:

- Platform vendors frequently release updates, patches and service packs which results in updating the platform as well. Hence, the platform used during compatibility tests may not be same as actual application implementation platform.
- Range of compatible platforms should already be defined in requirement statement. It is not possible or feasible to make the application compatible with all the platforms in the world.
- It is important to do cost-benefit analysis to understand the target market as it is not feasible to do exhaustive compatibility with all platforms if target market does not need it.
- Certain assumptions have to be made as the list of required platforms and configurations may not be complete.
- Platform configuration impacts the application performance to some extent.

Multiplatform Testing Process:

- Define the base platform for testing and do result analysis on base platform. Compatibility testing on different platforms is done and results are compared with base platform results. Any deviation in result is considered as defect.
- The expected platforms on which application compatibility is expected should be defined and approved by the customer beforehand.

Table 10.1 gives an example of multiplatform testing coverage for three platforms (say PA, PB, and PC) and a base/mother platform (say PM).

Table 10.1 Multiplatform testing coverage

Platform targeted	Distribution	Testing coverage
PM	-	Base platform, system testing
PA	80%	100%
PB	15%	25%
PC	5%	Smoke testing

- Assess test laboratory configuration with respect to base platform and target platform configuration mentioned in requirement statement. It must include the service packs or hot fixes to be included/excluded in testing and configuration of system which can impact the application.

- Usually, application interacts with the platform for performing various service tasks such as display and printing. The customer or designer must define the interfaces between application and platform. Extent of testing and specific test cases to be performed can be defined by list of interface items in the application that are being affected by platform change.

- Test cases selection and execution depends upon initial calculation of coverage offered or expected by customer, risk analysis, and cost-benefit analysis.

Types of Compatibility:

Friend Compatibility:

There is no change of application behavior between mother platform and new platform. The application utilizes all facilities and services available on the platform efficiently.

Neutral Compatibility:

The application behavior on new platform is similar to its working on parent platform. Only difference is that the application does not use the facilities provided by new platform at all. Instead, it uses its own utilities and services thus overburdening the system.

Enemy Compatibility:

The application does not perform as expected or does not perform at all on new platform.

10.4 SECURITY TESTING

Security testing ensures the security and protection of an application against unauthorized penetration. Every system has some vulnerable points easily prone to attacks. It is essential to protect the application from attacks which may put user's privacy, data and system at risk.

- Make a list of all possible penetrators of the system. It may include everyone who uses the system or accesses internet cloud through the system.

- Make a list of all penetration points which are susceptible to attacks easily.

- Make a penetration matrix with one dimension as perpetrators, and another dimension as a point of penetration. Each quadrant will give the possible point of failure of a system.

- Define the probability of security breakage, and impact of such breakage for each failure point represented by each quadrant. Each quadrant has probability and impact associated with it. Product of probability and impact represents the risk associated for the application. Higher score or greater RPN/RIN represents more problematic situation. Such cases must be handled on priority basis during development. They also indicate the probable areas where testing must be focused.

- Execute tests on the basis of high-risk prioritization which is a product of probability and impact or RPN/RIN.

Application Areas of Security Testing:

- Systems containing critical data of employees, customer and project may be prone to attacks. Such high confidential data of organisation must be protected.

- E-business and E-commerce systems involve financial transactions thus demand high protection of user's privacy.

- Communication system may be prone to information loss during transit. It may cause huge loss if perpetrators take/alter the information while in transit.

Security checking involves designing test cases that subvert the programs security checks and try to break the system defenses. Some of them are as follows:

- Users are not supposed to write or store their passwords anywhere. The passwords must not be shared with anyone. Obtaining a password using unofficial method to break a system is one way of security testing.

- People may be able to guess password using details like names of near and dear ones, birth date, etc. Such information can be used to decode the password.

- Login and password copying and pasting must not be allowed by the system. Small utilities can be used to break the password, if copy and paste functionalities are allowed.

- Idle terminals should get locked, after being left idle for some time. Timeframe may be decided by organisation. If the terminal does not get locked automatically, then an unauthorized person may start accessing the system as valid user has already logged in the system.

- Check permissions of different user groups and their privileges for system usage. Admin privilege is restricted to few people only.

- Direct backend changes in database should not be allowed. Otherwise, records are added in database without going through validation and verification procedure which may affect the application.

- There is limit defined for number of users for the system. To test it, create more users than allowed. Also, number of users with admin privilege should be restricted.
- System deleting admin account or one admin account deleting another admin account should not be allowed.
- Renaming admin group or deleting groups containing admin users or any active users should not be allowed.

10.5 PERFORMANCE TESTING

Performance testing is intended to check whether the system meets its performance requirements as mentioned in requirement statement. Performance criteria must be measurable. Examples:

- Adding a new record in database must take maximum 5 milliseconds.
- Searching of a record in a database containing one million records must not take more than 1 second.
- Sending information of say 1 MB size across the system with a network of 512 KBPS must not exceed 1 minute.

Testers should try above possibilities and verify the performance of the system. Generally, automated tools are used where strict performance requirements need to be tested.

10.6 VOLUME (LOAD) TESTING

Volume testing measures the maximum load a system can undergo before it collapses. As volume load increases, performance decreases. However, the goal is to find whether system crashes or survives due to increased volume rather than checking the performance.

Ways of Volume Testing:

- During design, set a limit on the number of users. This can prevent excess users accessing the system, thus protecting the system from any excessive load.
- Another way is to allow excess users to join but with a warning of slow response from system due to load. Sometimes, users may accept slow running system rather than a system which is unavailable.
- One approach is to create session timeout when user keeps system idle for a stipulated time.

Volume testing may require automation for stringent requirements as load is incremented by small factor and corresponding iterations are conducted to find the point of system collapse.

Examples of Volume Testing Process:

- First, transaction is given normal priority. Users are incremented each time and priority is also set to high till the system fails.
- User limit is set to 1000 concurrent users. As soon as 1001^{th} user tries to enter the system, it may not be allowed with a warning message. Another way is to allow but warn of slow performance.

10.7 STRESS TESTING

Stress testing defines the minimum resources the system will require for efficient and optimal performance. System performance deteriorates due to non-availability of resources. Stress testing also takes into consideration 'safety factor' to protect user from system failure due to resource constraints. To conduct stress testing, simple transactions are created and the system resources such as processor size, RAM, etc. are reduced to find the minimum level of resources till the system works.

10.8 RECOVERY TESTING

Recovery testing intends to find out how the system as a whole or an individual machine/server as a component of the entire system recovers from a disaster. There are various ways of disaster recovery and system requirement should specify the methods to be used during recovery.

10.8.1 System Recovery:

System is composed of several components such as machines, terminals, routers, and servers. System failure means failure of any of these components or failure of all or many of these components or communication failure between these components. There are three different ways of system recovery:

System returns to the point of integrity after encountering disaster:

When system encounters disaster, it is expected to return to the last action that the system knows before disaster occurrence. This last point of integrity differs from system to system and it is defined in requirement specification as per user expectations. Example: In some systems, 'refresh' button is often used to bring the system to previous page before it got lost. However, certain secured systems like IRCTC portal comes to the initial point which is login page. This affects user's experience as entire information of a transaction is lost and needs to be entered again. This way is low level disaster recovery.

Storing data in temporary location:

When system encounters disaster, the completed transactions till that point are stored in temporary memory. As the system returns to its original state, user is asked to confirm the commitment of stored transactions. User views them and if it accepts, the system resumes from that point. If user

rejects, transactions are permanently deleted and user begins from initial point. In this method, memory space is required to store transactions and space requirement increases with increase in users. This is an advanced way of handling disaster.

Completing the transaction:

When system encounters disaster, the transactions till that point are automatically committed. The system tries to complete the transaction till the last point in every possible way. This is midway amongst the two above ways to handle disaster.

10.8.2 Machine Recovery:

Machines such as application server and database server contain vital information and data. It is essential to protect this data in case of disaster. There are four ways to safeguard data:

Cold Recovery:

- Back up data at a defined frequency, such as once in a week on external device like tape/CD. When disaster occurs, recovery is done by loading this back up data on new machine and this machine replaces original machine. However, last updated data is permanently lost as backup is done at specified intervals only.

Warm Recovery:

- Backup is taken from one machine to another machine directly in this recovery.
- Frequent and automatic backup is possible. Data is already on the hard drive of both machines, if the backup is successful.
- In case of disaster, backup machine in introduced in the system temporarily.
- Backup machine configuration may differ from original machine. Up gradation of backup machine is required otherwise user faces problem with inferior backup machine.
- Maintenance cost of two machines is involved.
- Data may be lost in case data is updated after last backup.

Hot Recovery:

- In this recovery scenario, both, original and backup machines are present in the system.
- One machine is primary and another is backup machine. Both have similar configuration and capabilities.
- In case of disaster, control is shifted from primary machine to backup machine.

- Backup frequency is defined as per backup plan.
- Data may be lost in case data is updated after last backup.

Mirroring:

- Similar to hot recovery, there are two machines with same configuration.
- Data is backed up online and backup frequency is mirroring data, i.e., every millisecond data is copied from one machine to another.
- If primary machine fails, backup machine takes over without any manual intervention.
- Huge cost is involved but data availability is almost continuous.

10.9 INSTALLATION TESTING

Most of the applications need to be installed before using them. Installation testing is intended to find out how the application can be installed using installation guide or documentation provided.

Aim of Installation Testing:

- To check if the installation process is documented correctly in the user manual. Also, user must be able to easily interpret the instructions in manual.
- To give training to users, if installation demands the same. Users should be able to install application independently after requisite training.
- To provide up gradation document, if installation requires switching of systems or migrating from old machine to new one. Installation must be able to detect current configuration and guide users through up gradation accordingly.

Installation Process:

- Installation may need certain prerequisites for installation to complete. This should be detected during installation process and informed to user.
- Installation may be done using network, or from one machine to several machines at a time. It can also be done through devices like CD, pen drive, and floppies etc.
- Sometimes, remote installation is also required.
- According to requirements specification, installation process may be auto-run or semi-automatic or manual. For auto-run, installation is automated without any manual intervention. For semi-automatic, installation halts for required user inputs. For manual installation, users are trained to install the application.

Uninstallation Testing:

Uninstallation testing is used when requirement specification mentions the need of uninstallation. In this testing, all components and files of application are cleaned such that no thread of that application is left. The process is as follows: Image of hard disk will be captured before software installation. It will note all the files existing before installation. Then, tester installs application and again captures the hard disk image. These two images are compared and they should match exactly to prove clean uninstallation.

Upgradation Testing:

Software applications require upgradation to newer versions by installing updates released from product manufacturer from time to time. It is required that the application should already exist on the disk before upgradation. During upgradation, installer must be able to identify the older version of application already available on disk. If this existing application is more updated than the available upgradation, then upgradation should not be done. Upgradation is similar to installation process. It may be automatic, semi-automatic or manual. It may be done using CD, floppy disk, etc. User must be given adequate support in case of any problem in upgradation process.

10.10 REQUIREMENT TESTING

Requirement testing ensures that the system meets user expectations. Objectives of requirement testing are:

- Ensure correct implementation of user requirements.
- Processing adheres to organization and customer policies and standards.

Method of Requirement testing:

- Create requirement traceability matrix to determine whether all requirements are implemented or not during software development. Traceability includes requirements, designs, coding, test cases, and test results. Test case failure indicates requirement not met.
- Use checklist to verify whether system meets organisational policies, regulations and legal requirements.
- Create prototype or models to understand requirement characteristics.

10.11 REGRESSION TESTING

Regression testing intends to determine whether the changed components have introduced any error in unchanged components of the system. Regression testing can be done at:

- Unit level to ensure unit level changes do not affect other units of the system and unit fulfils its intended purpose.
- Module level to ensure module behavior is not affected by changing of individual units.
- System level to ensure system is performing correctly according to requirements even after changes made in some system parts.

Regression testing is performed when there is high risk that changes made in one part of software may affect unchanged components or system adversely. It is done by rerunning previously conducted successful tests to ensure that unchanged components function correctly after change is incorporated. It also involves reviewing previously prepared documents to ensure they contain correct information even after changes have been made in the system.

Methodology Where Regression Testing is used:

- Maintenance activities like defect fix, enhancement, reengineering, etc.
- Iterative development methodologies.
- Agile development methodology.

There are many tools available for automated regression testing. Automation is faster and cheaper alternative for regression testing as regression testing requires huge efforts and time.

10.12 ERROR HANDLING TESTING

When using application, user may face error due to wrong options selection or wrong data entered. Application is expected to guide users through error messages. Error message indicates that something wrong has occurred and how to resolve the issue and prevent errors. Error handling testing is done to determine:

- System ability to prevent users from entering erroneous data. Users should be alerted with error messages, in case of wrong entry or processing detected by the system.
- Application must recognize all expected errors and error messages should be displayed when any such error is encountered. Message must guide user through identification and error correction.
- Procedures must detect and correct high-probability errors before system processes such data.

Error messages may be of several categories:

- Preventive messages -System prohibits the user from entering wrong data. Example: If user tries to enter date ‘13/05/1989’ in the system which accepts date in ‘MM/DD/YYYY’ format, system detects and

prevents the user from entering date by displaying message that 13 is not a valid month.

- Auto-corrective message –System informs user about wrong entered data and automatically corrects it. This saves user's time and efforts but it can be problematic if system does not know the correct version of data.
- Suggestive message –System informs user about what is wrong and provides suggestions to correct the data. It is up to user to accept or reject suggestion. Automatic correction is avoided but user must know the correct entry of data and either repeat the transaction from beginning or accept the suggestion and correct wrong data.
- Detective message –System informs user about what is wrong but does not guide about how to correct it. No suggestions or guidance is provided. User solely has to do correction without any guidance or information about error correction.

Error handling must happen throughout development cycle. Good error handling assists user while working with system and improves user experience. Error correction must not introduce any error in system. Types of errors and strategy to handle them must be mentioned in requirement statement. Error handling cost and time estimation to correct it must be considered while developing system.

10.13 MANUAL SUPPORT TESTING

Manual support testing is intended to test the interfaces between user and application. It determines whether:

- Manual support procedures are sufficiently documented, complete and available to user. It can be in the form of online help or user manual or trouble shooting manual.
- Manual support staff is adequately trained to handle various conditions including entering data, processing, taking outputs as well as handling errors. Users are able to work with system independently.
- Manual support and automated segments are interfaced properly within the application.
- When users need assistance, provide input to manual support group and enable its entry into system at correct time.
- Prepare output reports and inform users to take necessary actions based on these reports.

10.14 INTERSYSTEM TESTING

Newly developed system may have to work with already existing systems which may be automated or manual. Intersystem testing tests interfaces

between two or more systems to ensure they work correctly together and support information transfer amongst them.

It determines whether:

- Parameters and data are correctly passed between systems.
- Documentation must be complete and accurate with expected inputs and outputs from the system.
- System testing is conducted each time when there is change in parameters, application, or other external systems.
- Manual verification of documentation is done to understand the relationship between different systems.

10.15 CONTROL TESTING

Control testing is done to check data validity, file integrity, audit trail, backup and recovery, and documentation for the system under development. It determines the following:

- Data processed is accurate, complete and usable. If mandatory fields are placed, then control must stop at those fields until they are filled by the user.
- Authorized transactions must enter system by validating user permissions. Unauthorized persons should not be allowed to access records or modify them.
- Audit trail is maintained to know transaction history made by users. Transactions must be traceable from start to end.
- Process must support requirement statement and meet user needs.
- Ensure data integrity and transaction processing from start to end.
- Identify risks associated with users using the system and accordingly grant access privileges.
- Create risk conditions to devise control mechanism. Expose the system to these conditions during testing to validate control mechanism.
- Evaluate effectiveness of controls as defined in requirement statement. Controls must be effective, efficient and usable.

10.16 SMOKE TESTING

Smoke testing intends to check whether the application is working or not. It basically aims to check if user can use the application. It is not any approval testing method. Smoke testing is conducted without any user inputs. Tester does installation, navigates through the application, and checks output by invoking certain functionalities. If smoke testing fails,

the application is rejected. Success of smoke testing is basic criteria for further system testing. Smoke testing is generally conducted by test manager or senior tester. It is also termed as ‘smell test’ as test manager may have to make judgment about the system in short period of time.

10.17 ADHOC TESTING

Adhoc testing is performed without any formal test plan, test scenario, or test data. It is also termed as ‘exploratory testing’ or ‘monkey testing’ or ‘random testing’. Testers use their previous testing experience and test the system functionalities. Sometimes, hidden defects are revealed which were not caught in earlier testing.

Advantages of adhoc testing:

- There is no particular sequence of testing. Test scenarios are adhoc which may cause stress to system. This may reveal defects which could not be identified while testing with a plan.
- Some test scenarios might be completely new which were never tried in earlier testing phases.
- It may require less time as there is no need to write any test plan or devise test cases. It is also called ‘playing with an application’.

Disadvantages of adhoc testing:

- Since it is random testing without any plan or steps written, tester may forget the steps by which the defect was identified and how the tester reached that point making the entire effort and time completely useless.
- Undefinable adhoc scenario intends to break the system which actually does not represent any real-life business scenario. Sometimes, the probability of occurrence of these events might be next to ‘0’.
- Adhoc testing require testers with knowledge and experience.

10.18 PARALLEL TESTING

Parallel testing compares the results, analyses the similarities and differences between newly designed system and existing system. It is done extensively in acceptance phases, typically in beta testing or business pilot where existing system, legacy system or manual operations are compared with the new system being developed. Parallel testing is also termed ‘comparison testing’ where new system behavior is compared with existing system whose behavior is considered correct. It is used extensively in compatibility testing. Parallel testing is done to determine whether,

- New version of application or new system performs correctly with reference to existing system that is supposed to be working correct.

- There is consistency or inconsistency between two systems. Consistency in terms of user interaction, user capabilities, transaction processing and controls designed for validation is also evaluated.
- Outputs of both systems are compared by providing same inputs to both.
- Both systems are used in parallel for certain time period for thorough comparison.
- Security, productivity, and effectiveness of new system must be comparable with the old system. If there is any lacuna in new system with respect to old system, new system may get rejected.

10.19 EXECUTION TESTING

Execution testing intends to ensure that system achieves desired level of proficiency in production environment when normal users are using it in normal circumstances. It may be considered as alpha testing if it is done in development environment, or beta testing if it is done in user environment. Data used in execution testing must be shared by customer. Execution testing involves actual working on the system in production environment to determine whether,

- System fulfils its design objectives as mentioned in requirement statement and expected by users. It evaluates user's experience while interacting with new system.
- System must be used at that point of time when results can be used to modify system structure, if required. Alpha and beta testing anomalies may be used to modify system, if required.
- Execution testing may be conducted by using hardware/software monitors, by simulating the functioning of the system, and by creating programs to evaluate system performance.

10.20 OPERATIONS TESTING

Operations testing are performed to check whether the operating procedures are correct as documented in user manuals, and staff can properly execute the application by referring the documentation. Operations testing determine,

- Completeness of operator documentation, user manual, etc. Users must be able to work with the system easily and correctly by referring these documents.
- If user training is required and after training, users are able to use system or not.
- Operations testing are done prior to placing the application in production environment. Actual users must be capable of working with system smoothly, and system must respond to users efficiently.

- Operators should perform testing without any assistance, as if it was part of normal computer operations.

10.21 COMPLIANCE TESTING

Compliance testing intends to check whether the application/system is in accordance with the prescribed standards, procedures and guidelines applicable to them as per domain, technology, customer, etc. There are many standards available for applications depending on the domain in which the application will work. These standards may be mandatory or recommended by customers, governing bodies, etc. There may be few regulatory or statutory requirements enforced by different agencies. Examples of standards applicable for software may include medical standards such as 'FDA (Food and Drug Administration) regulation' for software in medical domain in United States, standards for war equipments for software working in military operations and equipments, and special standards for software working in aviation industry. Compliance testing helps to determine whether,

- Development and maintenance methodologies are followed correctly or not while developing/maintaining system belonging to different domains. Domain related protocols must be followed.
- System documentation is complete and adheres to applicable standards.
- Compliance depends upon management's desire to have standards enforced as well as geographical and political environment where application will be working. Customer requirements must include definition of regulatory and statutory requirements.

Compliance testing is performed using following:

- Checklist prepared for evaluation or assessment of product
- Peer reviews to verify that the standards are met
- SQA reviews by quality professionals
- Internal audits

10.22 USABILITY TESTING

Usability testing intends to check 'ease of use' of an application to user. It provides user manual or help manual (including online help) to users. It determines whether,

- Application is simple, easy to use and understand by testing look, feel and support available can be used effectively or not.
- Application is easily executable through user interface. If training is required, it must be provided to users.

- Usability testing is done by,
- Direct observation of people using the system, noticing user interaction and noting their experience. In case of any problem, invoking help or referring user manuals to check if they are sufficient to help users in solving the problem.
- Conducting usability surveys by checking the deployment or implementation of system in production environment. Users must be able to use system on their own without any assistance.
- Beta testing or business pilot of application is user environment.
- Usability testing checks the following:
- Whether system outputs such as printouts and reports are meaningful or not.
- Is error diagnostic straight forward and easy to understand. Error messages must be simple and guide users correctly.
- Is user interface aligned with user requirements and standards imposed by regulatory bodies?
- Is the application easy to use for common users?
- Is there an exit option available at all points so that users can exit the system at any moment?
- System must not annoy user due to function unavailability, slow speed or bad user interface. System icons and pictures must be used appropriately for ease of users.
- System taking control from user without indicating when it will be returned can be a problem as user might not be aware of estimated time it will take. System must indicate current operation going on and estimated time taken for that operation.
- System must provide online help or user manual for self service. In case of any problem, users must be able to invoke help.
- Whether system is consistent in its function and overall design.

10.23 DECISION TABLE TESTING

Decision table is a good way to capture system requirements that contain logical conditions, and to document internal system design handling various conditions faced by the system. They may be used to record complex business rules that a system is expected to implement. Specifications are analyzed, and conditions and actions of the system are identified. The input conditions and actions are most often stated in such a way that they can either be true or false (Boolean).

Decision table contains triggering conditions, often combinations of true and false for all input conditions, and the resulting actions for each combination of conditions. Each table column corresponds to certain business rule that defines a unique combination of conditions, which result in the execution of the actions associated with that rule. The coverage standard commonly used with decision table testing is to have at least one test per column, which typically involves covering all combinations of triggering conditions. One may use equivalence partitioning and boundary value analysis for testing such system.

Table 10.2 shows a sample decision table.

Table 10.2 Decision table

Purchased volume and discounts				
No. of Books	1-50	51-500	501-5000	5001 and more
Discount offered	0%	2%	3%	5%

The strength of decision table testing is that it creates combinations of conditions that might not otherwise have been exercised during testing. It may be applied to all situations when the actions of the software depending on several logical conditions are occurring at same time. Table 10.2 indicates single dimension of variable. In practical conditions, multiple dimensions are possible.

10.24 DOCUMENTATION TESTING

Documentation testing involves review of all the documentation delivered to the customer along with the application. When system is delivered to customer, it is not only the executable or sources which are delivered but it should contain all these support documentations required for future maintenance of a system. System documentation may contain requirement specifications, requirement changes, impact analysis, design artifacts, design changes, impact analysis, code documentation, project plans, and test plans. All these documents are needed to build the right system. Sometimes, documentation also includes release notes, known issues and limitations if any, installation guide, user guide, and troubleshooting guide.

The purpose of documentation testing is to thoroughly go through all written material that will be presented to user as a part of implementation. Testing is done using different ways:

- Ask users to follow all documented procedures. These should be written to provide step-by-step guidance on how to accomplish a given task. If the users cannot successfully complete the procedures, then documentation should be improved.

- Have people with strong language skills to review all the documentation for professionalism and readability.
- Try out all alternative ways mentioned in document to accomplish a task. In many cases, the primary way works, but the alternatives do not work as expected.
- Documentation should strictly be aligned with company policies and these policies and standards must be reviewed and approved by appropriate authorities in the organization.
- Ensure the accuracy of any manual forms, checklists, and templates.

10.25 TRAINING TESTING

Sometimes, the vendor is expected to provide training to end users who will be using the system in future. Hence, training is also tested as a part of system testing. This implies that the training must be ready at this point in the life cycle rather than waiting for the end of project. Testing must be performed in a controlled test environment to ensure that it is effective, accurate, and bug-free.

In case of distance learning, ensure that correct technology serves the purpose. There should be adequate equipment's for imparting training. Training material must be deliverable. It should be tested thoroughly before delivering it.

10.26 RAPID TESTING

Rapid testing is used when there is less time to obtain full test coverage using conventional methodologies. It can be used to complement conventional structured testing. It finds the biggest bugs in the shortest time, and provides the highest value for money. In an ideal world, rapid testing would not be necessary, but in most development projects, there are a number of critical times when it is essential to make an instantaneous assessment of the products quality at that particular moment.

Areas where rapid testing is applied extensively are:

- 'Proof of concept' test early in the development cycle.
- Prior to, or following migration from development to the production environment.
- Preparing development milestones to trigger funding or investment.
- Prior to public release or delivery to the customer.

Although most projects undergo continuous testing, it does not usually produce the information required to deal with the situations listed above. Usually, testing completes just prior to launch, and conventional testing techniques often cannot be applied to incomplete or often changing software.

Structured testing is a vital part of any development project but it cannot meet all the quality assurance objectives. Its primary objective is usually affirmative testing, i.e., to verify that the software does all the things it is supposed to do. However, software is now so complex that there are often a seemingly infinite number of permutations of the data variables, and variations in the time domain can add another layer of complexity.

In addition to affirmative testing, it is necessary to identify undesirable behavior, so that it can be corrected. But structured testing is far less useful as a technique for doing this. The number of permutations means the time required for analysis, scripting and execution. This required time may not be available and time can be reduced if the tester had idea about the likely faults.

Rapid Testing process:

Rapid testing is based on exploratory testing techniques, which means that the tester has a general test plan in mind but is not constrained by it. The plan can be adapted dynamically in response to previous test results. A skilled tester can quickly find defects which escaped a scripted test. Exploratory testing is also called 'expert testing', as it requires a high level of technical knowledge and experience. However, rapid testing does not guarantee complete test coverage.

Rapid testing extends the exploratory concept by making judgment about what faults to report and the level of details to be recorded. Once a fault has been identified, the time taken to investigate and document it reduces the time available to find other faults, so the tester may fail to find the serious faults if they spend too much time reporting trivial issues. This is called 'quality threshold' and it is fundamental to the effectiveness of rapid testing. Many testers find it an alien concept but it is a necessary one.

Initial quality threshold might be determined by consulting with customer, but it may vary during the course of testing depending on the product quality.

10.27 CONTROL FLOW GRAPH

Control flow graph is flow of control during program execution. If a program does not have any decision, flow goes in single direction. If decision is taken by a program, then different flow graphs are possible. Each decision induces multiple paths in a program while it is getting executed.

Dominators:

There exists a set of code in the program which will always be executed when a path is selected. This is termed as 'dominator'. These paths are independent of any decision or branch.

Post-Dominator:

When end of application is reached from any point, there exists a set of code that is always encountered, which is termed as ‘post-dominator’. There may be several possible paths depending upon the number of decisions.

10.27.1 Program Dependence Graph:

Program execution depends on two factors: data dependence and control dependence.

- Data dependence flow depends on input data. Example: Loop statements ‘if’, ‘while’ where data decides loop will be executed or not.
- Control dependence flow is the flow of instructions due to control points in a program. If the program has redundant code, control will never go to it. Control may be defined by user requirements.

10.27.2 Category Partition Method:

Category partition method is used to generate the test cases from requirements. Following steps are involved in generating test cases by category partitioning method:

Analyze Requirements:

Requirements are categorized into functions, user interface, and performance requirements. They are placed in different test cases to be tested independently.

Identify Categories:

Inputs producing specific categories of output are analyzed. Decision tables can be used to identify such categories.

Partition Categories:

Respective input-output pairs are placed into different partitions.

Identification of Constraint:

There may be some input output categories which cannot exist together. They must be identified and excluded from testing.

Creating Test Cases accordingly:

Test cases are created keeping in mind the possible constraints defined in user requirements.

Processing the Test Cases:

The test cases defined above are executed and results are captured.

Evaluate the Output:

The output is verified by comparing it with expected output.

Generate Test Sets:

If obtained output and expected output match, then it is added in test set.

10.27.3 Test Generation from Predicate:

A condition which a code can achieve is called 'predicate'. It indicates a condition and its action. Condition is represented by predicate while action represents expected result of code execution based on the condition. Predicate testing tests such predicate to detect any fault created by developer while coding a program.

10.27.4 Fault Model for Predicate Testing:

Predicate testing may target for three different classes of faults as defects: incorrect Boolean operator used, incorrect relational operator used, and incorrect arithmetic operator used.

10.27.5 Difference between Control Flow and Data Flow:

Table 10.3 Difference between control flow and data flow

Control flow	Data flow
Control flow is process oriented. It defines the direction of control flow as per system decision.	Data flow is information oriented.
It does not manage data or pass data from one component to another.	Data flow passes data from one component to another.
It functions as a task coordinator. Control flow requires task completion.	Data transformation occurs simultaneously or sequentially.
It is synchronous in nature. Unrelated tasks can be synchronized.	It may or may not be synchronous in nature.
Tasks can be executed in parallel or one after another.	Generally, tasks are executed one after another, if there is serial dependency. Otherwise, they are independent of each other.

10.28 GENERATING TESTS ON THE BASIS OF COMBINATORIAL DESIGNS

The environment in which an application will work consists of various configurations of elements like hardware, browser, operating system, etc. During testing, it may not be possible to test all configurations of above

elements. Certain factors are provided of the environment in which application will work. Each factor may be tested using one test case, which increases the total number of test cases which can go infinity. In combinatorial designs, we try to reach some finite level of test cases which can give adequate confidence that application will execute in all possible combinations. The set of input and output are partitioned such that selected value represents each partition.

10.28.1 Combinatorial Test Design Process:

Modeling the Input Space and Test Environment:

The model consists of set of factors and corresponding levels. Factors are decided on the basis of interaction between application and environment in which it is expected to work.

Generate Combinatorial Object:

This is an array of factors and levels selected for testing. Such an array will have each row covering at least one test configuration from the list.

Generate Tests and Test Configurations:

From the combinatorial object, a tester may have to generate the test cases and test configurations. Figure 10.1 shows schematically how the test cases can be generated.

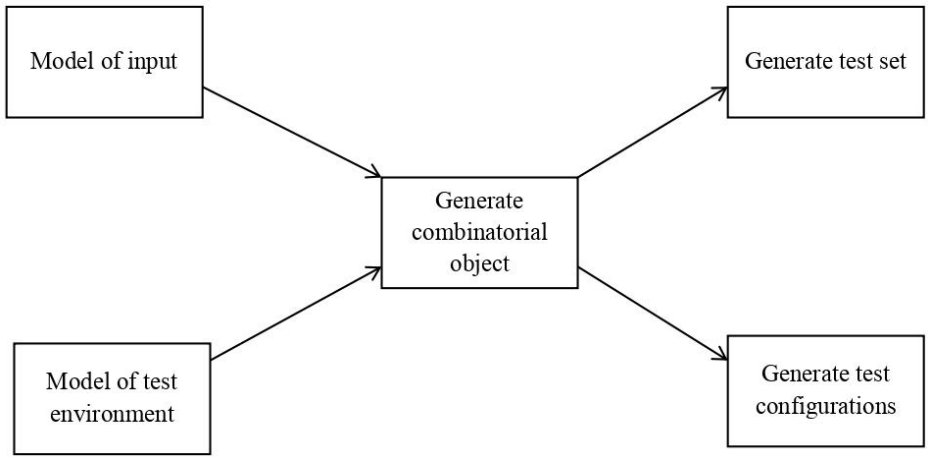


Fig. 10.1 Process of generating test and test configurations using combinatorial designs

10.28.2 Generating Fault Model from Combinatorial Designs:

The aim of combinatorial designs is to generate test cases which have maximum probability to uncover faults or defects. Simple fault are triggered by individual input variable irrespective of other input variables. Pairwise interaction faults are triggered when there are two variable combinations which can yield a fault. Three-way interaction fault is when three variables in combination create a fault but individually there is no fault.

10.29 STATE GRAPH

State graph and state table are useful models for describing software behavior under various input conditions. State testing approach is based upon the finite state machine model for the structures and specifications of an application under testing. In a state graph, states are represented by nodes. The system changes its state when an input is provided. This is called ‘state transition’.

With multiple state graphs, drawing them and understanding becomes difficult. To remove this complexity, state tables are used. State tables may be defined as follows:

- Each row indicates a state of an application.
- Each column indicates the input provided to an application.
- Intersection of rows and columns indicate the next state of an application.

10.29.1 Characteristics of Good State Graphs:

- Total number of possible states is equal to product of all possibilities of factors that make a state.
- For every state and output, there is exactly one transition specified to exactly one state.
- For every transition, there is one output, however small it may be.
- For every state, there is a sequence of inputs that will drive the system back to the same state.

10.29.2 Number of States:

Number of states is an important parameter to determine the extent of testing. Some factors affecting number of states are:

- All component factors of state
- All allowable values of the factor

Thus, number of states is a product of number of allowable values of all the factors.

Impossible States:

States which appear to be impossible in reality due to some limitations outside the application in real life scenario.

Equivalent States:

When transitions of two or multiple states result into the same final state, they are called ‘equivalent states’. Equivalent states can be identified as,

- The rows corresponding to two states are identical with respect to inputs/outputs/next state of an application.
- There are two sets of rows which have identical state graphs.

Unreachable (Redundant) State:

When the requirements are captured or designs are made, some states are created which fall in a category where control will never reach. They differ from impossible states in the sense that the flow will never reach to them. Impossible states are not practical but unreachable states are redundant.

Dead States:

Dead state is a state where there is no reversal possible. These states cannot be exited even if the user wishes to come out of them. There is no possibility to come back to the original state.

10.29.3 Matrix of Graphs:

Matrix of graph is a square array with equal number of rows and columns. Each row and column represents one node.

- Matrix size corresponds to number of nodes.
- There is scope to show direct relation between every node to any other node.
- Connection from node 1 to node 2 may or may not have reverse connection.

A graph contains a set of nodes and relations between them. If 'A' and 'B' are two nodes and 'R' is the relation between them, it is represented as A-R-B. Different types of relations which may exist are as follows:

Transitive Relationship:

Consider a relation between three nodes 'A', 'B', and 'C' such that A-R-B and B-R-C indicate A-R-C, and then it is called 'transitive relationship'.

Reflexive Relationship:

Reflexive relationships are self-loops.

Symmetric Relationship:

Consider a relationship between nodes 'A' and 'B' where A-R-B indicates that there is B-R-A. This is considered as symmetric relationship or undirected relationship.

Equivalence Relationship:

If a relationship between two nodes is reflexive, transitive and symmetric, then it is called 'equivalence relationship'.

10.30 RISK ASSOCIATED WITH NEW TECHNOLOGIES

Organizations adapt to new technologies due to several benefits of these technologies. New technology may mean a technology which is completely new to the world or not new to the world but new to the organization. In any case, these new technologies introduce some risks along with benefits. Risks may be faced by the development team as well as end users. Following are the different risks:

1. Unproven technology:

New technologies are used by organization due to their advantages. But, it may have some drawbacks or lacunae, which the organization might not be aware of. Also, it may not be possible to use all advantages offered by the new technology. Sometimes, the new technology is not assessed properly for positive and negative factors before using it. Technology needs some time to mature and as users use it, they provide feedback to technology owners, and thus help in the maturing process.

2. Technology itself is defective or not mature enough to use:

The new technology may be defective as there may have been lot of limitations and unknown aspects overlooked by the group releasing it. Some aspects may have been compromised and people using the new technology may not be aware of the areas where one should be careful. It may introduce some defects in the product, and users may find it difficult to use such products, or may face several problems while using such applications. New technology needs sufficient validation along with verification and fixing of the defects found during these processes.

3. Technology is inefficient as it is not matured:

The new technology may be inefficient, and may not have all the expected services. It may hamper performance and usability of an application adversely. There may be limitations which the development team might not be aware of thus surprising the end user.

4. Technology is incompatible with other technologies already in use:

Introduction of new technologies may interfere with existing system's working or new systems cannot communicate with existing systems. This may pose major problem as users do not wish to discard existing systems so easily while adapting to new systems. Data transfer and communication between different systems is a basic customer requirement and customer can even reject new systems if there is no connectivity/communication offered with existing applications and technologies.

5. New technology makes existing technology obsolete:

While new technology may have several benefits, customer may not like to scrap existing systems to completely replace them with new systems.

6. **Variation between technology delivered and documentation provided:**

Every technology comes with documentation like user manuals and troubleshooting manuals. Users and developers refer this documentation to understand the usage of new technology. If a gap exists between documentation and actual technology working, then people may not be able to use the technology by referring the documentation. Thus, documentation not being in sync with technology may hamper usage of new technology as people may not know how to use it.

7. **Lack of developer's/user's competencies to use new technology:**

New technology may require people to undergo training to understand and develop capabilities to use it. Some skills may be acquired by classroom training; some may be acquired by hands-on experience with mentor such as on-the-job training while some may need special training. New technology must be user friendly so that people with basic knowledge can use them with little guidance or trial and error approach. If new technologies are difficult at all fronts, users may dislike them and may be reluctant to use them.

8. **Lack of knowledge and skill for optimal technology usage:**

Technology may not be usable by organization if it does not have people with required skills and experience. There may be different options to maximize the technology usage, but people might be unaware of these options due to lack of knowledge or skill.

9. **Technology is not incorporated into organization's process definition:**

Developing a good product requires a mix of good people, good technology and good process working together. If organizational database does not have the definition of processes which can support new technologies, then success depends upon the heroics of people and chances. This may lead to major problems with these technologies, and people may try to implement technology with their own methods which may not be the best approach.

10. **Technology leads to obsolescence of existing development/testing tools:**

Development tools/test tools available with an organization may not be able to support new technologies, if there are compatibility issues. In such cases, we may have to rely on manual efforts of development and testing which may be incapable or insufficient for the purpose. New technology may need new tools which mean further investment and training for developing organization/users.

11. Inadequate support for technology from vendor:

When an organization adapts new technology, it relies on the vendor providing service support for such technology, at least for initial phases of usage. Service support may include trouble shooting and trainings which may not be available as and when required by organisation/users. Many aspects depend upon people, both internal as well as external, to make the project successful. Technology may become a problem if there is not adequate or completely unavailable vendor support.

10.31 PROCESS MATURITY LEVEL OF TECHNOLOGY

Organisations may have different process maturities, and usage of new technologies may be affected by these maturities. Maturity levels may be defined as follows:

Level 1: Adhoc Usage of New Technology:

This is an initial or base maturity level for given technology. In this maturity level, organisation uses technologies based on available people skills and expertise. Some organisations specialize in particular technology, and each and every project is done in that technology irrespective of whether it is latest one or not. People often select technology as per their usage convenience and not as per project requirement.

Level 2: Managed Usage of New Technology:

Technologies are selected based on evaluation done by users by referring support materials provided by the vendor or some suggestions and experience. An organisation works with new technology based on such descriptions without having any hands-on knowledge on technology. This is ‘static approach of technology selection’ or ‘evaluation based upon documentation’ where everything goes according to documentation without ever concerning the technology usefulness and its drawbacks.

Level 3: Defined Usage of New Technology:

An organisation selects a technology on the basis of documentation and conducts experimentation to complete the project using that technology. Sometimes, results are not sufficient but once technology is selected, it cannot be changed. Lot of research is done to make selected technology successful. The organisation may learn lessons from its success (or failures) and the same can be used for other projects in similar technology. Making mistakes and correcting them is acceptable in this approach.

Level 4: Quantitatively Managed Usage of New Technology:

At this level, the benefits and limitations of selected technology for a project are completely measured. Customer/user is involved in making

decisions about the selection of technology on the basis of benefits and shortcomings analysis.

Level 5: Optimized Use of Technology:

At this highest maturity level, organisation has a plan to overcome the shortcomings faced by particular technology and enhance its benefits to give better results. Customer does not suffer due to limitations of technology. Drawbacks are resolved to optimize and enhance technology usage.

10.32 TESTING ADEQUACY OF CONTROL IN NEW TECHNOLOGY USAGE

Following controls are essential while testing new technologies:

1. **Testing actual performance achieved as against stated performance of the technology by manufacturer:**

Vendor may claim certain performance parameters while releasing new technology. These claims may include different advantages/services available while using new technology. Tester must check the actual performance standards, verify and validate all the claims made regarding the technology. If any gap is found between the claims made and actual performance, then it is termed as defect or shortcoming. In Level 3 maturity, such limitations limit the benefits offered by the technology and a claim may be lodged with technology vendors, if feasible. At Level 5 maturity, these limitations are overcome by some alternative approach while claim may be lodged with vendor. Testers must ensure the following:

- Documentation provided with technology must be correct as it represents actual technology execution. Any defects in documentation can cause usability issue.
- Training courses regarding new technology must be effective and complete. This can be validated by users.
- New technology must be compatible with existing technology so that customer may retain old applications along with new one. Compatibility issues, if any, must be addressed.
- Claimed performance must match with actual performance.
- Promised vendor support must be actually offered when user needs it. Service levels must be defined and achieved in contract.
- New technology is tested using new test processes and tools. It may add to customer delight if existing tools can be used as it is or with some modifications to do testing of new technology.

2. Test the adequacy of current process definition available to control technology:

An organisation needs process definition to support new technology implementation and usage. If support is unavailable, then testers may have to raise issues and get the process definition. It can be a risky situation and customer must be involved in decision making or customer must be informed about the possible lacunae of process definition. If the processes are available, their usage must be tested. Testers must ensure that,

- Development standards and usage are available for the selected technology.
- Procedures for development, maintenance, troubleshooting and usage are available.
- Assured quality control with verification and validation is available.

If there are inadequate controls, following actions may be initiated by testing group:

- Identify risks associated with technologies not supported by process framework and report them to stakeholders. It may be shared with user groups.
- Identify potential weak areas and create mitigation action plan, if risk turns into reality.
- If problems are discovered during testing, they must be resolved and retesting should be done.

3. Assess adequacy of staff skills to effectively use technology:

People must be adequately trained and skilled to use the technologies effectively and efficiently. Organisation must take actions to improve the skills or procure people with desired skills. Testers must ensure that,

- Technological process maturity level of an organisation is assessed and known to organisation leadership. Actions must be initiated to achieve optimizing level.
- Training is provided to developers and testers for using new technology efficiently and effectively. Effectiveness of training must be evaluated.
- Performance evaluation of technologies on the basis of limitations and benefits must be conducted. Users must be informed about possible shortcomings of the given technology.

10.33 OBJECT-ORIENTED APPLICATION TESTING

Object-oriented development has made a dramatic change in development methodologies. One object may be used at several instances giving a

benefit of optimisation, reusability and flexibility. It improves productivity with good maintainability of an application. There are many effective approaches to test object-oriented software. A test process that complements object-oriented design and programming can significantly increase reuse, quality and productivity of development and testing process.

- Object-oriented applications may also face simple programming mistakes, anticipated interaction and incorrect or missing behaviour of application or individual object at different instances. Reuse in no way guarantees that a sufficient number of paths and object states have been exercised to reveal all application faults. Reuse is limited to the extent that supplier classes are trustworthy and can be used at several places during application development.

- Simple and sequential test process beginning from unit test, then integrating all units and performing integration testing and then system testing may not be relevant to object-oriented development. Owing to iterative and incremental nature of object-oriented development, tests need not be limited to boundary and scope. They can be designed and exercised at many points in development process. Thus "design a little, code a little" becomes "design a little, code a little and test a little".

- Adequate testing requires sophisticated understanding of the system under testing. One must be able to develop abstract views of the dynamics of control flow, data flow and state space in a formal model used for development. One must have complete understanding of system requirements and be able to define the expected results for any input and state selected as a test case.

- There are many interactions among components that cannot be easily foreseen in unit testing until all system components are integrated and exercised through testing. Even if we eliminate all individual sources of error, integration errors may arise in object-oriented development. Compared to conventional systems, object-oriented systems have more components which must be integrated and tested for correctness individually and as a group. Since there are certain system elements that are not present until code is loaded and exercised, there is no way that all faults could be identified and eliminated by class or class-cluster testing alone, even if every method was subjected to a formal proof of correctness and unit testing. Static methods cannot reveal interaction errors within different objects with the expected performance issues in real-time systems.

- Testing activities can begin and proceed in parallel with concept definition, object oriented architecture, and object oriented designs and programming through integration and system testing. When testing is correctly interleaved with development, it adds considerable value to the entire development process as defects are detected as soon as they occur.

- The cost of finding and correcting errors increases with increase in time between fault injection and detection. The lowest cost is possible when one prevents errors from entering the system. If a fault goes unnoticed, it can easily take hours or days of debugging to diagnose, locate and correct it. Failures in operational systems can cause severe secondary problems including customer complaints, rejection of application, etc. Proper testing is feasible as compared to 'find and fix' defect randomly.

- Effective testing also provides information about likely sources of defects. Object-oriented features like polymorphism, inheritance and encapsulation produce more error opportunities as compared to conventional languages. Testing strategy must assist testers to fetch for these new kinds of errors and set some criteria when fetching must complete.

- Each sub class is a new and different context for an inherited super-class feature. Different test cases are needed for each context. Inherited features must be exercised in the context of sub classes. Inherited methods should be retested even if they were not changed in individual instance. Sometimes, methods work perfectly in superclass, but their action and interaction may be affected in sub classes which must be tested.

- Even if many server objects of a given class work correctly at top level, there is a probability of a client class using it incorrectly. Thus, all uses of a server object need to be exercised at client classes also.

10.34 TESTING OF INTERNAL CONTROLS

Introduction of software system is associated with risks and failures too. Testers must do risk assessment as follows:

- Inherent and residual risk acceptable to user/customer using the software. Inherent risks are born risks of particular approach while residual risks are the risks left after adequate controls are provided.

- Estimating probability of occurrence of particular event in terms of percentage. Also, if user gets idea about risk approaching, then how much reaction time the user gets before risk materializes.

- Qualitative and quantitative measurements of probability, impact and detection ability of different risks associated with application usage can give a risk rating expressed as RPN or RIN. Risks must be prevented or corrected, as the case may be.

- Correlation of events where occurrence of one risk may increase or decrease the probability or impact of some other risk must be known. Such risks together can create more problems than individual occurrence or may eliminate the problem.

10.34.1 Testing of Transaction Processing Control:

An application may process the data received from different inputs and provide outputs in different forms. Application controls must be designed to ensure data accuracy in all phases of data receipt, processing and output. The control placement may include the following:

Transaction Origination:

The points where the data originates must be controlled. Sometimes, data originates in one system and it is transferred to given system for processing through different methods. Data may originate in manual operations also. Generally, these controls may not be applicable to the given system, if data preparation happens outside the system either manually or automatically but data entry must be controlled in such cases.

Transaction Entry in System:

The point where the data enters into the system for further processing must be controlled by the application as it is the most vulnerable point. There must be verification and validation of data entries at entry point and user must be supported adequately through help option and proper error messages should be displayed for common users.

Transaction Communications within/outside the System:

When the data is communicated from one place to another either in same system or between different systems, adequate precaution must be taken so that neither the data gets lost nor modified. Data transfer amount must be measured and matched before leaving the system and when it comes back into database.

Transaction Processing by the System:

When the system processes data, it must have adequate control to check completeness of processing. All the data entered into system must be processed, and outcome must contain results from all processed data. User must be notified if any part of data is not processed completely or partially.

Storage and Retrieval of Data from the Database:

Data may be stored on different media and databases and also physically or logically at different locations. Data entering in the system and outputs generated from the system must match with each other. Stored data should not be altered or deleted.

Transaction Output:

Processed output may be transferred from one system to another or may be delivered to users in different ways including printing, displaying or sending data from one location to another or from one system to another. Transaction output must show the processing results correctly and data input and output must be matched in data transfer process.

10.34.2 Testing Security Control:

Security controls are designed to protect the system from internal or external attacks from possible penetrators. Security testing is a thought process where one needs to understand the thinking of an intruder and try to device control mechanisms accordingly. Following analysis should be done:

Points where Security is more likely to be Penetrated (Points of Penetration):

These are the areas where the system is exposed to outside world and there are possibilities of outside attacks. At these points, attacker's entry probability is high. These are also called 'threat points'. Few threat points may be,

- Data preparation stage which can be a manual operation or it may occur in some other system.
- Computer operations where the system processes or transfers data from one place to another. It may include server rooms, processors, etc.
- Non-IT areas where people working with system do not understand system security concept. They may be ignorant of security practices and may cause failure unknowingly.
- Software development, maintenance and enhancement phases where data is used for building, maintaining or testing of an application.
- Online data preparation facilities where data is not validated adequately before entering the system.
- Digital media storage facilities which may be subject to electro-magnetic fields, temperatures, humidity, etc. Storage facilities may not be secured enough and storage media may get damaged due to wrong handling.
- Online operations where data is manually entered in system without any verification and validation or data is transferred from one system to another without any control.

Points where System is Least Protected (Vulnerable points):

It is not possible to protect the system completely. Certain points will be left where the system is least protected which are called 'vulnerable' or 'weak' system points.

Attributes of Effective Security Control:

A tester may test the following attributes to find the effectiveness of security control:

- **Simplicity of Control and Usage:** Controls must be simple to **understand and use for the users.**
- **Failure Safe Controls:** Controls must be safe and free from failure.
- **Open Design for Controls:** Control design must be flexible enough to accept technology and process changes to modify controls accordingly.
- **Separation of Privileges of Users:** System must enforce separation of user privileges to avoid any unauthorised entries bypassing the controls.
- **Psychological Acceptability of Controls by Users:** An organisation must plan for adequate training for users. Security and controls and their purpose must be well explained to people so that they can psychologically accept the controls and do not bypass them.
- **Layered Defense in System:** Entire system must not collapse. If any system part fails, there must be alternative defense mechanism to detect and prevent complete system failure. If unauthorised transaction entry occurs, it must be detected and proper actions must be initiated.
- **Compromised Recording:** Transactions made by users must be recorded and audited. This assists in early problem detection in transactions and processing. However, it may hamper user's privacy and should be incorporated if requirements specify the same. Audit trail is an example of compromised recording in a system.

10.35 'COTS' TESTING

'COTS' stands for 'Commercially Of The Shelf' software. These are readily available software in the market which user can directly buy and use. They can be integrated in a new development or used as a tool for development or testing activities.

Why Software Organisations Use Cots?

The most common question that arises is why would a software development organisation buy and use software from outside? There are several reasons given below:

Line of Business:

An organisation's line of business might differ from the software that it requires.

Example: Organisation developing software for banking domain is not in a line of business to develop automation software required for testing. In such cases, organisation might purchase and directly use this testing software from market without investing resources and time for creating it.

Cost-Benefit Analysis:

Sometimes, it is very costly to develop software in-house due to various constraints like lack of knowledge, skills, resources, budget, etc. Buying the software becomes more convenient and cost effective than creating it. Usually, 'COTS' products are much cheaper than the projects as cost is distributed over number of users.

Expertise/Domain Knowledge:

An organisation might know usage of software with no knowledge of its creation. Sometimes, it is sufficient to have understanding of using software without going into details of its development. Organisation might purchase them simply from market and use it.

Delivery Schedules:

'COTS' are available immediately in market by paying for them. This saves the time and efforts of development. This aspect is beneficial when efforts and schedule do not justify use and benefits. Cost of purchasing software may be less compared to developing it.

Features of Cots Testing:

Testers must remember following features while testing a 'COTS' product:

- 'COTS' are aligned with general requirements and market trends. It might not exactly match organisation's needs and expectations. Analysis and percentage fit of 'COTS' to the organisation business should be done to decide its success ratio.
- Some 'COTS' require changing business processes to suite the 'COTS' implementation in organisation. This is another way of Business Process Reengineering (BPR) for an organisation where internationally proven practices can be implemented by using 'COTS'. 'COTS' may have some of their best processes accepted at national/international level and organisation may get benefitted by using such processes along with the product.
- Sometimes, 'COTS' may need configuration of software or system to suite business needs. Generally, when 'COTS' are implemented, many business rules must be defined to customize the software to the organisation.

Challenges in 'Cots' Testing:

Some of the challenges faced by testers while testing 'COTS' are as follows:

- Requirement statement and designs may not be available to testers as product manufacturer never shares them with any customer. While buying an operating system, we cannot expect that the organisation will give requirement statement for such product. Generally, in

normal testing process, test scenarios and test cases refer to requirements and design, but in case of 'COTS', business requirements are referred which may differ from product requirements.

- Verification and validation records prepared during SDLC are very important for system testing and acceptance testing. SDLC acceptance reduces the risk of buying a wrong product. But in case of 'COTS', these records might be unavailable to testers before performing acceptance testing. Tester must do software testing by understanding organisation's perspective to find out whether 'COTS' should be accepted or rejected. Code reviews, requirement reviews, design reviews, unit testing, and integration testing records are not available to testers.

There are two methods of conducting acceptance testing before buying 'COTS'.

Evaluation of Software Product:

- Evaluation of 'COTS' is done by referring information available about it through various sources like white papers, user manuals, peer experiences, expert's judgement, available demos and internet searching.

- It is a static measurement of software to understand its features and suitability for business as there is no hands-on or self-experience of using such product. Decision of buying 'COTS' depends on the available documents and information.

Assessment of Software Product:

- Assessment includes executing test cases on the product. Many product manufacturers give evaluation versions with limited time period validity so that users can get hands-on and decide about purchase. Expectation is the buyer must conduct testing and decide to buy or not.

- It is a dynamic measurement to gauge the product suitability. There must be test plan, test scenario, and test cases for product testing. Understanding of product requirements and attributes is essential.

- Assessment may need basic understanding of particular type of software and assessment skill to operated and assess new software.

'Cots' Test Process:

Assure Completeness of Need Specification:

Testers must be completely aware of the organisational requirements before testing 'COTS'. Requirements may be from various areas as expressed in 'TELOS' (Technical, Economic, Legal, Operational and System).

- Expected output formats and reports essential for organisation must be defined beforehand. Some 'COTS' can be configured to get desired output formats while some may not deliver such reports and formats. They may need some external customization.
- Information needed for management's decision making must be defined beforehand. Percentage fit may be decided based on such definitions which may be used while deciding to buy software or not.
- There may be some statutory/regulatory requirements applicable to the organisation and COTS must help in achieving them. If COTS does not satisfy these requirements, organisation may have to devise a method to enhance COTS with external programming or else reject it.

Define Critical Success Factor of Buying:

Testers must understand organisation's motive to purchase COTS. Critical success factors are those which define whether COTS has fulfilled its intended use or not. Some of them may be:

- Ease of use
- Scalability
- Cost effectiveness
- Portability
- Reliability
- Security

Determine Compatibility with Environmental Variables:

An organisation must have definition of working environment where 'COTS' will be implemented. Environment may include hardware, software, and people around it. 'COTS' must fit in the existing environment as there may be several systems already existing in the specified environment. Organisation may not change its environment for implementing any COTS unless there is no way out for it. COTS compatibility may include the following aspects:

- Hardware Compatibility: Environment comprises of machines, servers, printers and communication devices that exist and used by the organisation before introduction of COTS. It is expected that COTS must work without any problem in this environment.
- Operating System Compatibility: Operating systems, browsers and databases used before COTS implementation should be usable after its implementation. Data transfer from one system to another is usually avoided due to data compatibility issues. Usage of same database improves data processing.

- Software Compatibility: There may be existing software in system where COTS will be implemented. COTS should not affect their existence or working. Other software may provide input or accept output from COTS software. COTS must be able to integrate with them and communicate effectively.

- Data Compatibility: Data transfer may happen from one system to another during COTS implementation. COTS should not hamper or interfere with data, format and transfer process. Data format, style, and frequency mismatch can cause severe system problems.

- Communication Compatibility: Protocols used in communication must be compatible to prevent any communication loss. If there is no compatibility of communication protocols, adopters will be required to convert protocols to facilitate communication.

Assure that COTS can be integrated with Business:

- Manual systems may be partially or completely replaced by COTS software in an organisation. This may affect employees count and they may need to upgrade skills to use COTS.

- Existing processes, methods, forms, formats and templates used by existing manual system may be replaced in COTS implementation. These changes may affect users, auditors, etc. which should be taken into consideration while deciding to implement COTS in an organisation.

- There may be increase or decrease in number of steps and sub processes due to COTS and users must be comfortable with such changes. Increase in processes may be resisted by users if they find it unnecessary.

Demonstrating COTS in Operation:

COTS is also subjected to alpha and beta acceptance testing similar to other projects. Vendor site demonstration of COTS represents alpha testing and actual usage site demonstration represents beta testing.

- **Demonstration at Vendor site:** These may be done using sample data provided by vendor. Objective is to provide basic awareness to user about the software. It is used to provide training to users about new product.

- **Demonstration at Customer site:** These may be done using customer supplied data or real time data. Users are involved in demonstrations and experts may guide them to use new product. Users may get hands-on experience under expert supervision. This also exposes users to software and gives basic training to use it. Users understand advantages and limitations of software, if any, during such demonstrations.

Evaluate People Fit:

Testers need to analyse whether people would be able to work with the system effectively or not. There should be understanding whether software can be used as it is or may require some configuration, modification or external changes to make it usable. Sometimes, additional training and support may be required by users.

10.36 CLIENT-SERVER TESTING

Client-server is an initial improvement from stand-alone applications where there are several clients communicating with the server. There are many advantages of client-server over stand-alone application. Client-server architecture gives an opportunity to many users to work with software simultaneously. Client server system may be viewed as-requests coming from many clients and server is serving these requests.

Features of Client-Server Application:

- Client-server system comprises of **clients** connected to **server** across a **network**.
- Client and server are connected by real connection.
- Multiple clients use the system at a time and they can communicate with the server. Sometimes, clients may communicate with each other via server.
- Server is aware of client configuration beforehand.

Testing Approach of Client-Server System:

Client-server testing involves component and integration testing followed by various specialized testing as per scope of testing involved.

Component Testing:

One needs to define the approach and test plan for testing client and server individually. One may have to devise simulators to replace corresponding components to test the target component. For server testing, client simulator may be needed and client testing may require server simulator. We may have to test network by using client and server simulators at a time.

Integration Testing:

After successful testing of clients, servers and network individually, they are brought together to form the system and system test cases are executed. Client server communication testing is done in integration testing. There are regular testing methods like functionality testing and user interface testing to ensure that system meets the requirement specifications and design specifications correctly. In addition to these, several special testing techniques are as follows:

Performance Testing:

System performance is tested when many clients communicate with server at the same time. Similarly, volume testing and stress testing may also be used. Since number of clients is already known to system, we can test the system under maximum load as well as normal load. Various user interactions may be used for stress testing.

Concurrency Testing:

It may be possible that multiple users may access same record at a time. Concurrency testing is required to understand the system behaviour under such circumstances.

Disaster Recovery/Business Continuity Testing:

When the client and server are communicating with each other, there exists a probability of communication break due to various reasons or failure of either client or server or link connecting them. This testing is used to understand system behaviour in cases of disaster. It may involve testing the scenario of such failures at different system points and actions initiated by the system in each case. Requirement specification must describe the possible expectations in case of any failure.

Testing for Extended Periods:

In client-server applications, generally, server is never shut down unless there is some agreed (Service Level Agreement) SLA where server may be shut down for maintenance. It may be expected that server is running 24 * 7 for extended period. Testing is conducted over an extended period to understand if service level of network and server deteriorates over a time due to some reasons like memory leakage.

Compatibility Testing:

Client and server are subjected to different networks when used by users in production. Servers may be in different hardware, software or operating system environment than the recommended one. Clients may significantly differ from the expected environmental variables. Testing must ensure that performance is maintained on the range of hardware and software configurations and users must be adequately protected in case of configuration mismatch. Similarly, any limiting factors must be informed to prospective user.

Other testing such as security testing and compliance testing may be done, if needed, as per scope of testing and type of system.

10.37 WEB APPLICATION TESTING

Web application is further improvement in client-server applications where the clients can communicate with servers through virtual connectivity. It has many advantages over client-server application as multiple server networks can be accessed at a time from the same client. It

improves communication between people at different locations significantly.

Features of Web Application:

- Number of clients connecting to a server is very large. Sometimes, number may tend to infinity. Client configurations cannot be controlled by definition.
- Different clients may have different configurations and server may be unable to communicate directly with clients. To resolve this, a universal client 'browser' is required in such circumstances.
- Communication protocols may differ from system to system. Sometimes, adopters are required to convert these communication protocols and make them compatible.
- Client and server are connected through World Wide Web cloud and there is no direct physical connectivity. As they are connected virtually, communication needs address where it is expected to reach, from client to server as well as from server to client.
- One client may be able to connect to several servers at a time. This helps in faster communication between different domains.
- Generally, there is no installation needed at client other than browser. Nearly all work is done by the server.

Testing Approach of Web Application:

Web application involves component testing, integration testing, functionality testing and GUI testing followed by various specialised testing.

Component Testing:

One must define the approach and test plan for testing web application individually at client side and at server side. One may have to design simulators to replace corresponding components. During server testing, client simulator will be needed and vice versa. Network testing is also required.

Integration Testing:

Successfully tested servers and clients are brought together to form the web system and system test cases are executed. Communication between client and server is tested in integration testing.

There are other testing ways like functionality testing and user interface testing to ensure that system meets the requirement specifications and design specifications correctly. In addition to them, several other testing is involved as follows:

Performance Testing:

System performance is tested as huge numbers of clients communicate with server simultaneously. Similarly, volume testing and stress testing is done to test these applications. System requirements specifications should define maximum and normal load conditions so that they can be tested. Simulators are used extensively for such testing.

Concurrency Testing:

It may be possible that multiple users access same record simultaneously. Concurrency testing is needed to understand system behaviour under such circumstances. Probability of concurrency increases with increase in number of users.

Disaster Recovery/Business Continuity Testing:

When the machine is communicating with the web server, there exists a possibility of communication break due to several reasons like connectivity failure, client failure and server failure. Testing for disaster recovery and business continuity involves testing the scenario of such failures and actions taken by the system in each case. Requirement specification must describe the expected system behaviour in case of such failures. MTTR (Mean Time to Repair) and MTBF (Mean Time between Failures) are very important tests for web applications.

Testing for Extended Periods:

In case of web applications, generally, the server is never shut down. It is expected to run 24 * 7 over extended time period and there must be a provision of alternate server (hot recovery/mirroring) if requirements specify the same. Testing is conducted over an extended period to understand if service level of server deteriorates over a time due to some reasons like memory leakage.

Security Testing:

As the communication occurs through virtual network, security is important. Applications may use communication protocols, coding and decoding mechanisms, and schemes to maintain system security. System must be tested for possible weak areas called ‘vulnerabilities’ and possible intruders trying to attack the system called ‘penetrators’.

Compatibility Testing:

Web applications may be placed in different environments when users use them in production. Servers may be exposed to different hardware, software or operating system environment than the expected one. Client browsers may differ significantly from the expected environmental variables. Testing must ensure that performance is maintained on the range of hardware and software configurations and users must be adequately protected in case of configuration mismatch. Similarly, any limiting factors must be informed to the prospective user.

10.38 MOBILE APPLICATION TESTING (PDA DEVICES)

Today's generation uses pocket devices extensively for communication and computing due to mobility offered by them. There is tremendous increase in memory levels and technologies adopted by such appliances. With the advent of Bluetooth and Wi Fi, many PDA's are replacing desktop computers due to ease of usage. New technologies have converted normal communication device into internet-driven palm tops.

Testing Limitations of PDA's:

Scenarios and test cases designed for web applications may not be applicable to mobile applications. Mobile applications exhibit different behaviour and usage pattern. There are few limitations on PDA due to following factors:

- Hardware and software used on PDAs differ significantly due to lack of standardization, and varied manufacturer's preferences. Usability testing on PDAs should take into consideration variations in hardware, software and configurations.
- Memory available in PDAs is limited as compared to desktops.
- PDAs have limited usability even after invention of touchpad and joystick. Normal desktops usage is more convenient owing to keyboard, mouse along with touch screen.
- Data input has many limitations as single key may mean different inputs depending upon the number of times it is pressed. There is limitation for providing help and maintaining operability of PDA due to its small size.
- Battery life may be limited in PDAs thus limiting its lighting availability.
- Bandwidth available with Bluetooth and Wi Fi in PDAs is much less as compared to bandwidth availability by other means like optical cables.

Interface Design of PDAs:

- The smaller size and resolution of the PDA screen presents usability challenges to testers. Reading from the screen and scrolling up and down is inconvenient.
- Instructions and other text must be used sparingly and carefully as they occupy screen and require scrolling. Instructions cause content to be pushed below the fold which can be missed by users.
- Links must be concise and contain only necessary keywords. Generally, specific options must be presented before general options to improve usability.

10.39 E-BUSINESS/E-COMMERCE TESTING

With changing business scenarios and technology advancements, e-Business/e-Commerce applications have become popular. There is a small difference between e-Business and e-Commerce application. While e-Commerce concerns mainly with money transactions, e-Business concerns with all business aspects including money, advertising, and sales.

Distinct Parts of e-business:

- **Information Access** by the common user is very important from business point of view. All the products available with enterprise must be accessible to users who wish to buy them with product information. Quick links may be provided to reach the area of interest easily and quickly. Grouping items into some categories and carting also eases access.

- **Self-Services** must be available for users where they can select item, quantity and make online payment accordingly.

- **Shopping Services** including advertisement, carting and billing are completely online and users should be able to execute transactions smoothly just like they do it in a real shop.

- **Interpersonal Communication Services** can be used to inform users about the billing amount or inform shopkeeper about stocks availability and demand trends. It can be used to give more information about the products available for sale.

- **E-Business** represents a virtual enterprise where all the actions are done by users through internet which replaces physical shopping and transactions.

Testing Approach for e-business/e-commerce:

- Software applications should be user friendly so that people can perform transaction easily and safely. Usability, performance and reliability are critical success factors for such applications. Users may reject applications due to poor performance and business is directly impacted. Consistency in delivering results is important.

- Regulatory and statutory requirements should be strictly followed as their violation is a legal offence.

- Security and privacy is critical in e-Business and e-Commerce applications and these may be enforced as statutory requirements. Personal information and transaction information of users must not be disclosed to any third-party attack. Anonymity should be maintained.

- Non-functional requirements such as system performance, user interfaces, and online help are essential in addition to functional requirements. Application's look and feel, working speed, and

disaster recovery ability are important aspects from user's perspective.

E-commerce Quality Challenges:

- Stringent quality standards are associated with e-Business sites. Usage experience for users must be positive. Otherwise, people would hesitate to use site impacting business as there is no physical store in existence.
- If the visitor experience is negative due to application's slow response time, crash, and privacy violations then consumer confidence deteriorates and consumers would not perform transaction thus impacting business.

E-business/E-commerce Development:

Development of e-Business/e-Commerce applications may not follow waterfall or iterative model. Agile methodologies and spiral development models are used extensively. It is characterized by the following:

- Rapid and easy assembly of application modules is essential as the system size increases as defined by spiral methodology. Initially, some parts of an application are delivered and used, and as per user response and expectations, latter parts are then delivered at multiple instances.
- Testing of component functionality and performance at each increment is required to ensure correct integration. Huge regression testing cycles may be required.
- Since there may not be real time testing, simulation is done by designing models to simulate real world scenario and then testing is performed.
- Usually, such applications are deployed in a distributed environment, 24 * 7 * 365 for an extended period, and there is no downtime allowable. MTTR and MTBF are important parameters for assuring service to users.
- Monitoring performance and transactions over an extended period is essential to understand if there is any deterioration of service level over a time span.
- Analysing system effectiveness and gathering business intelligence may be essential to maximize sale and profit.

Incorporating Legal Standards:

Statutory and legal requirements are one of the most important aspects of e-Business and e-Commerce applications. These requirements must be thoroughly understood and implemented in an application. Any violation of these requirements is punishable under law. Also, they keep changing

from time to time as per policy and strategy decisions of government and categories of industry that they belong to.

- All critical business functions with respect to common users are identified and evaluated on the basis of their criticality.
- Mechanisms must be in place to ensure connectivity and handling connection loss. Disaster recovery procedures and business continuity procedures must be developed.
- Systems are tested to assess security of online transactions. User information must be protected and privacy practices must be applied stringently.
- Privacy audits must be conducted to confirm correct working of strategies to protect customer privacy and confidentiality.
- Vulnerabilities are analysed to prevent hacker attacks and virus attacks. Users may not visit site if there are possible vulnerabilities and threats.
- Disaster avoidance measures are developed such as redundant systems, alternative routing, precise change controls, encryption, capacity planning, load and stress testing, and access control.

10.40 AGILE DEVELOPMENT TESTING

'Agile development' is popular in software development models. Agile development is based on twelve agility principles mentioned in agile manifesto. Agile models prioritize clients and focus more on procedures, quality delivery and flexibility in embracing dynamic requirements rather than being static to strictly follow guidelines and documentation.

Agility Testing:

There are various agile process models such as scrum, extreme programming, feature-driven development, test-driven development etc. Test plans are made to suit the model adopted and purpose. Also, the underlying agile principle of delivering working software at faster speed should be considered.

Critical Points for Agile Testing:

- **Competencies/Maturity of Agile Development and Test Team:**

When working with agile principle, it is important to have teams, customer and management who psychologically accept agile approach. It involves dynamic ability to accept change quickly, deliver good working product as soon as possible, and communicate effectively and efficiently with team members as well as stakeholders. Agile implementation prefers generalist approach as compared to specialist approach. It needs people

with high maturity and technical competence to adapt to changing customer needs.

- **Development and Test Process Variability:**

Every process has an inborn variability. One may have to attack the generic reasons of variations while there may be some controls to identify special causes of variations.

- **Change Management and Communication:**

Change is inevitable in agile. There must be strong communication amongst development team, test team, customer, and other stakeholders to adapt to changing scenario. Requirement change at any point in development process must be welcomed and everyone must come together to give best service to customer.

- **Test Process Flexibility:**

Different test plans are made to test software. However, as the process begins, many changes may be required. Test plans should be flexible enough to adapt to these changes.

- **Focus on Business Objective:**

There is always a pressure to deliver software faster in agile development. Time pressure may come from stakeholders or development. It is important to focus on business objectives while defining test processes, adapting to change, and handling pressure. Cost-benefit analysis may be done to deal with defect fix and software release. It is impossible to find all defects but aim should be to maximize user protection from any accidental failure. Testing should achieve both extremes.

- **Stakeholder Maturity/Involvement:**

Agile process demands maturity from stakeholders and all the people involved since there is time pressure, adapting to dynamic requirements, and faster delivery pressure.

10.41 DATA WAREHOUSING TESTING

A data warehouse is a repository of an organisation's electronically stored data. Data warehouses are designed to facilitate reporting and analysis. The classic definition of data warehouse focuses on data storage. However, the means to retrieve and analyse data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehouse system. The advantage of using data warehouse is that a data analyst can perform complex queries and analysis (data mining) on the information within data warehouse without slowing down the operational systems.

Data Warehouse Definition:

Data in data warehouse is subjected to few definitions:

Subject-oriented:

Subject-oriented data warehouses are designed to help the user in analysing data. The data is organised so that all the data elements relating to the same real world event or object are linked together.

Integrated:

Database integration is closely related to subject orientation. Data warehouses must place data from different sources into a consistent format. The database contains data from most or all of an organisation's operational applications and is made consistent.

Time-variant:

The changes done to data in database are tracked and recorded to produce reports on data varying with time. In order to discover trends in business, analysts need large amounts of data. Time-variant is data warehouse's track of changes made in data over time.

Non-volatile:

Data in the database is never over written or deleted once committed-the data is static and read-only but retained for future reporting. Once data enters warehouse, it should not change as the purpose of data warehouse is to be able to analyse what has occurred.

Benefits of Data Warehousing:

- Provides a common data model for all data of interest regardless of its source. This enables easy reporting and analysis as compared to retrieving information from multiple data models.
- Prior to loading data into data warehouse, inconsistencies are identified and resolved. This greatly simplifies reporting and analysis.
- Information in the data warehouse is under the control of data warehouse users so that even if the source system data is purged over time, information in the warehouse can be stored safely for extended time period.
- Data warehouses are separate from operational systems, hence; data retrieval does not slow down operational systems.
- Data warehouses facilitate decision support system applications such as trend reports, exception reports and reports that show actual performance versus goals.

Testing Process for Data Warehouse:

Testing for a data warehouse includes requirements testing, unit testing, integration testing followed by acceptance testing.

Requirements Testing

The main aim for performing requirements testing is to check stated requirements for completeness. In a data warehouse, requirements are related to reporting. Hence, it is crucial to verify whether these reporting requirements can be catered using the available data. Successful requirements are structured closely to business rules and address functionality and performance expected by users.

Unit Testing:

Unit testing involves the following:

- Whether the application is accessing and selecting correct data from correct source as expected by users.
- All data transformations should be aligned with business rules and data warehouse should be populated with correct data.
- Testing the rejected records that do not fulfill transformation rules.

Integration Testing:

After unit testing completes, integration testing is performed to test initial and incremental loading of data warehouse. Integration testing involves the following:

Verify Report Data with Source:

Although the data present in a data warehouse will be stored at an aggregate level yet it must be compared to source systems. Test team must verify the granular data stored in data warehouse against the available source data.

Field Level Data Verification:

Test team must understand the linkages for the fields displayed in the report and must trace back and compare that with the source systems.

Creating Queries:

Create queries to fetch and verify the data from source and target. Sometimes, it is not possible to do the complex transformations done in ETL. In such a case, the data can be transferred to some file and calculations can be performed.

Data Completeness:

Basic test of data completeness is to verify that all expected data loads into the data warehouse. This includes validating that all records, all fields and the full contents of each field are loaded. This may cover:

- Comparing record counts between source data, data loaded to the warehouse and rejected records.
- Comparing unique values of key fields between source data and data loaded to the warehouse.
- Utilizing a data profiling tool that shows the range and value distributions of fields in a data set.
- Populating the full contents of each field to validate that no truncation occurs at any step in the process.
- Testing the boundaries of each field to find any database limitations.

Data Transformation:

Validating that data is transformed correctly from database is based on business rules. This can be the complex part of testing.

- Create a spreadsheet of scenarios of input data and expected results and validate these with the business customer.
- Create test data that includes all scenarios.
- Utilize data profiling results to compare range and distribution of values in each field between source and target data.
- Validate correct processing.
- Validate that data types in the warehouse are as specified in the design and/or data model.
- Set up data scenarios that test referential integrity between tables.
- Validate parent-to-child relationships in the data.

Data Quality:

Data quality rules are defined during data warehouse design. It may include:

- Reject the record if a certain decimal field has non-numeric data.
- Substitute null if a certain decimal field has non-numeric data.
- Duplicate records.

Performance and Scalability:

As data volume in a data warehouse grows, load times can be expected to increase and performance of queries can degrade. The aim of performance testing is to point out any potential weaknesses in the design such as reading a file multiple times or creating unnecessary intermediate files.

- Load the database with peak expected production volumes to ensure that this volume of data can be loaded within the specific time period. The time period may be defined in SLA.
- Compare these loading times to loads performed with a smaller amount of data to anticipate scalability issues.
- Monitor the timing of the reject process and consider how large volumes of rejected data will be handled.
- Perform simple and multiple join queries to validate query performance on large database volumes.

10.42 LET US SUM UP

In this chapter, we have seen impact of new techniques of software development on testing. This chapter also covers what is meant by new technology and its associated risks. It also deals with organisational process maturity with respect to evolving technologies. The testing approaches of new technology covered following methodologies and business applications:

- Object-oriented development
- Internal controls
- Commercially Of The Shelf (COTS) software
- Client server testing
- Web application
- PDAs/Mobile applications
- E-Commerce and e-Business
- Data warehouse systems

10.43 EXERCISES

1. Describe the risk associated with new technology usage.
2. Explain technology process maturity.
3. Explain the test process for new technology.
4. Explain the process of testing object-oriented development.

5. Explain the process of testing of internal controls.
6. Explain how transaction processing controls can be tested?
7. Explain the process of testing security controls.
8. What are the attributes of a good control?
9. Why software organisations buy ‘COTS’ software?
10. Explain COTS testing process.
11. What are the challenges in COTS testing?
12. Differentiate between evaluation and assessment of COTS.
13. Describe a process of client-server testing.
14. Describe the testing process for web application.
15. Describe the mobile applications (PDA) testing process.
16. Describe the critical quality issues of e-Commerce and e-Business.
17. Describe the process of testing data warehouse systems.

*****