T.Y. B.Sc.

(Computer Science)

SEMESTER - V (CBCS)

SOFTWARE TESTING AND QUALITY ASSURANCE

SUBJECT CODE - USCS503

Prof. Ravindra Kulkarni
Vice-Chancellor,
University of Mumbai,

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

Prof. Santosh Rathod
I/c Director,
CDOE, University of Mumbai,

Programme Co-ordinator : **Shri. Mandar Bhanushe**
Head, Faculty of Science and Technology
CDOE, University of Mumbai, Mumbai

Course Co-ordinator : **Ms. Mitali Vijay Shewale**
Doctoral Researcher,
Veermata Jijabai Technological Institute
Mumbai

Editor : **Mr. Anish Raut**
Manager,
Dahua Technology India Pvt. Ltd.

Course Writers : **Ms. Mitali Vijay Shewale**
Doctoral Researcher,
Veermata Jijabai Technological Institute
Mumbai

: **Mr. Palash Ingle**
Research Assistant,
Sejong University, South Korea

June 2024, Print - I

Published by : I/c Director,
Centre for Distance and Online Education,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed : Mumbai University Press
Printed by Vidyanagari, Santacruz (E), Mumbai - 400 098

CONTENTS

Course: USCS503	TOPICS (Credits : 03 Lectures/Week:03) Software Testing and Quality Assurance	
Objectives: To provide learner with knowledge in Software Testing techniques. To understand how testing methods can be used as an effective tools in providing quality assurance concerning for software. To provide skills to design test case plan for testing software **Expected Learning Outcomes:** Understand various software testing methods and strategies. Understand a variety of software metrics, and identify defects and managing those defects for improvement in quality for given software. Design SQA activities, SQA strategy, formal technical review report for software quality control and assurance.		
Unit I	**Software Testing and Introduction to quality :** Introduction, Nature of errors, an example for Testing, Definition of Quality , QA, QC, QM and SQA , Software Development Life Cycle, Software Quality Factors **Verification and Validation :** Definition of V &V , Different types of V & V Mechanisms, Concepts of Software Reviews, Inspection and Walkthrough **Software Testing Techniques :** Testing Fundamentals, Test Case Design, White Box Testing and its types, Black Box Testing and its types	**15L**
Unit II	**Software Testing Strategies :** Strategic Approach to Software Testing, Unit Testing, Integration Testing, Validation Testing, System Testing **Software Metrics :** Concept and Developing Metrics, Different types of Metrics, Complexity metrics **Defect Management:** Definition of Defects, Defect Management Process, Defect Reporting, Metrics Related to Defects, Using Defects for Process Improvement.	**15L**
Unit III	**Software Quality Assurance :** Quality Concepts, Quality Movement, Background Issues, SQA activities, Software Reviews, Formal Technical Reviews, Formal approaches to SQA, Statistical Quality Assurance, Software Reliability, The ISO 9000 Quality Standards, , SQA Plan , Six sigma, Informal Reviews	**15L**

	Quality Improvement : Introduction, Pareto Diagrams, Cause-effect Diagrams, Scatter Diagrams, Run charts **Quality Costs :** Defining Quality Costs, Types of Quality Costs, Quality Cost Measurement, Utilizing Quality Costs for Decision-Making	

Textbook(s):

1. Software Engineering for Students, A Programming Approach, Douglas Bell, 4th Edition,, Pearson Education, 2005
2. Software Engineering – A Practitioners Approach, Roger S. Pressman, 5th Edition, Tata McGraw Hill, 2001
3. Quality Management, Donna C. S. Summers, 5th Edition, Prentice-Hall, 2010.
4. Total Quality Management, Dale H. Besterfield, 3rd Edition, Prentice Hall, 2003.

Additional Reference(s):

1. Software engineering: An Engineering approach, J.F. Peters, W. Pedrycz , John Wiley,2004
2. Software Testing and Quality Assurance Theory and Practice, Kshirsagar Naik, Priyadarshi Tripathy , John Wiley & Sons, Inc. , Publication, 2008
3. Software Engineering and Testing, B. B. Agarwal, S. P. Tayal, M. Gupta, Jones and Bartlett Publishers, 2010

1

INTRODUCTION TO SOFTWARE TESTING

Unit Structure :

1.0 OBJECTIVES

After completion of this module, you will learn:

- Definition of software testing
- Types of quality control models.
- Software Development Lifecycle.
- Verification and validation, verification methods.
- Black box testing, white box testing

1.1 INTRODUCTION

In this chapter, we are going to explore the intriguing area of software testing, which is a key component in terms of developing software. Everyone might have questions such as what is a software and why it is so important to test a software? How should we test it and who should test it? There are enormous questions, and this section answers the questions.

We are going to see a simple test case, errors associated with software and how they can be minimized. The main purpose of software testing can be understood when a programmer develops a code, and nobody likes it when it gives errors. Such situations are implied on using software testing and make us think that is software reliable? A competent software tester must project credibility and possess the qualities of curiosity, persistence, creativity, diplomacy, and persuasion.

In this section, we will go over the fundamentals of software testing, demonstrate how a test case is written, investigate different types of errors,

and the importance of quality. This chapter builds a strong foundation for comprehending the topic of software testing.

1.2 SOFTWARE TESTING

This is the main topic of the chapter. From the name itself it is evident that we are talking about testing software. Some people might confuse it with debugging. However, debugging means just removing errors from your programs whereas testing helps to discover undiscovered errors.

The act of testing involves running software with the goal of identifying errors. In this text,

The very crucial questions of who should conduct the testing and when it should begin are addressed.

1.2.1 What is Software?

Software, often known as a program, is a set of instructions for a computer to carry out anexplicit task. Software is a generic term used to refer to applications, scripts and programs that run on a device. It is divided into application software and system software. Any software designed to perform only specific tasks is known as application software (word, PowerPoint, etc.) and system software is responsible for running a computer's hardware, manage the resources and make the applications run on it (operating systems, game engines, etc.).

In older times, software was developed for certain machines and sold alongside the hardware it operated on. Afterwards, software started being marketed on floppy disks in the 1980s, and then CDs and DVDs. Nowadays, most software is bought and downloaded simply from the internet.

1.2.2 Need for Software Testing

Machines and humans rely on software for achieving their goals related to a task. We use software on daily basis and some software which are available free on internet can be a major source of threat to computer systems.90% of people use software in their daily lives, making it a product that is widely accessible.Only after a software product has undergone a proper process of development, testing, and bug repair it should be released.

We should look for errors in the preliminary stages of software development. In comparison to problems that we might discover in the later stages of software development, the expense of fixing such errors will be quite affordable. According to Figure 1, the cost of fixing errors spikes from the specification phase to the test phase to the maintenance phase. The relative cost of fixing a fault depends on what stage of the software development life cycle it occurs at.

The longer you wait to address a defect, the more money it will cost to remedy it.

Software has a high failure rate due to the lack of importance given to quality and testing.Smaller businesses with little resources run the risk of failing if they don't pay enough attention to software quality and undertake adequate testing.This is precisely why testing is required for every software.It is important to keep in mind that software is primarily tested to make sure it complies with standards and fulfills consumer expectations.

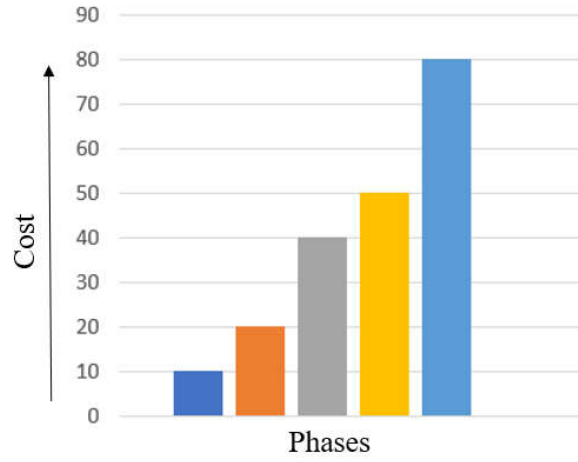


Figure 1.Statistics of cost of fixing errors

1.2.3 Who should test the software?

Software testing is not the responsibility of a single person, it is teamwork. It should come naturally to every member of a team to test the software. However, there is a specific group of people designated to undergo software testing known as 'Software Testers'. The issue here is that because the developers are so closely involved in the software's development, it is exceedingly challenging for them to identify mistakes in their own work.

The testers must be circumspect, inquisitive, critical but not judgmental, and effective communicators. They should have the courage to suspect any error and hand over the errors to the development team so that they can rework on it.

1.2.4 Qualities of a good tester

Testing is a crucial step in the software development process, and a good tester can reveal whether a product will succeed or fail. The tester must have the following qualities:

1. Honest and Intuitive:

A good tester should be honest about his work and should poses the expertise in testing process. He should be intuitive and proactive in situations which are unwelcomed. A good tester knows how to save time and identify which errors are to be solved first and which require less attention.

2. Explorer:

The testers can delve into unfamiliar environments and uncover bugs that would otherwise go unnoticed.Thanks to a little imagination and a willingness to take risks.

3. Good communicative skills:

Good software testers are gentle and skilled at telling the developers undesirable news.They approach the developers diplomatically, persuade them when necessary, and have their bugs repaired.

4. Learns from past mistakes:

Each error gives a source of improvement to the software, and this helps the tester to learn from it and apply the learning in future. A good tester takes the opportunity to learn and improve and always double checks their findings.

5. Organized:

Good testers are always very well organized. They prepare test cases step-by-step and use checklists, files, and statistics to support their conclusions. They always make sure to double-check their findings.

1.3 SOFTWARE QUALITY

Software quality means different to different people as it is highly context driven. An end-user always demands high-quality software. It covers several aspects such as good performance, proper specifications, meeting all the operational requirements, and efficiency. Good quality software helps save both time and money as it is less likely to have bugs which will free up time during testing and maintenance phases.

1.3.1 Quality Revolution

The roots of quality revolution can be traced back to 1950's, when there was a mass movement of people resulting in globalization. The global manufacturing quality was poor back then and William Edwards Deming who was an American engineer, statistician, professor, author, lecturer, and management consultant came up with the theory of constantly improving quality. His ideas first gained attraction with the Japanese manufacturing industry, which is why Japanese cars have been known for so long for superior qualityand reliability.He worked with various Japanese researchers on statistical quality control (SQC) methods. Measurements and statistics are the foundation of the field of statistical quality control. Instead of relying on intuition and experience, decisions and strategies are established based on the gathering and assessment of real data in the form of metrics.He described the plan-do-check-act (PDCA) cycle, also known as the Shewhart cycle (Figure 2). Setting goals, tying them to measurable milestones, and monitoring progress toward those milestones are the activities that make up the Shewhart cycle.

The idea to extend quality control beyond the manufacturing units and into the entire company was first forth by an American Joseph M. Juran in 1954. He emphasized the value of systems thinking, which starts with the requirements for the product, the design, the testing of the prototype, the appropriate use of the equipment, and the accurate process feedback. Juran encouraged Japan to transition from SQC to TQC (Total Quality Control). This includes quality control (QC), audits, the quality circle, and the promotion of quality management principles throughout the entire organization.

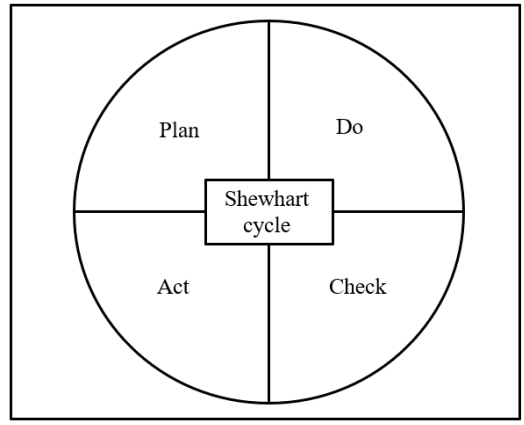


Figure 2.Shewhart cycle

Plan: Decide on the goal and the method for delivering the results.

Do: Put the strategy into action and evaluate its success.

Check: Measurements should be evaluated, and decision-makers should be informed of the findings.

Act:Make the necessary improvements to the procedure.

Quality became more and more valued as time went on.In 1968, Kaoru Ishikawa, developed a cause-and-effect diagram famously known as Ishikawa (Figure 3) based on TQC. He found that quality is affected by four common factors, materials, machines, methods, and measurements. Many Japanese companies maintain a detailed documentation of their quality efforts which led to widespread top-quality movement.

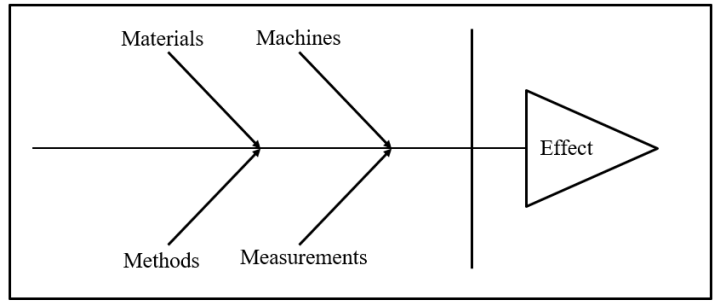


Figure 3. Ishikawa model

1.3.2 Software Quality models

Various software quality models have been proposed to define quality and its related attributes. There are many standards developed by a group of experts called the International Organization for Standardization (ISO). ISO 9126 is the most important standard which includes six characteristics of quality, functionality, reliability, usability, efficiency, maintainability, and portability.Carnegie Mellon University's Software Engineering Institute (SEI) developed the Capability Maturity Model(CMM). The CMM framework rates development processes on a scale of 1 to 5, which is commonly referred to as level 1 through level 5. Level 1 is the beginning point, whereas level 5, optimizedis the highest level of process maturity.The McCall's quality factors notion of software quality was put forth by McCall, Richards, and Walters in 1977. A system's behavioral feature is represented by the quality factor. The 11 quality factors are: correctness, reliability, efficiency, integrity, usability, maintainability, testability, flexibility, portability, reusability, and interoperability.

1. **Characteristics of ISO 9126**

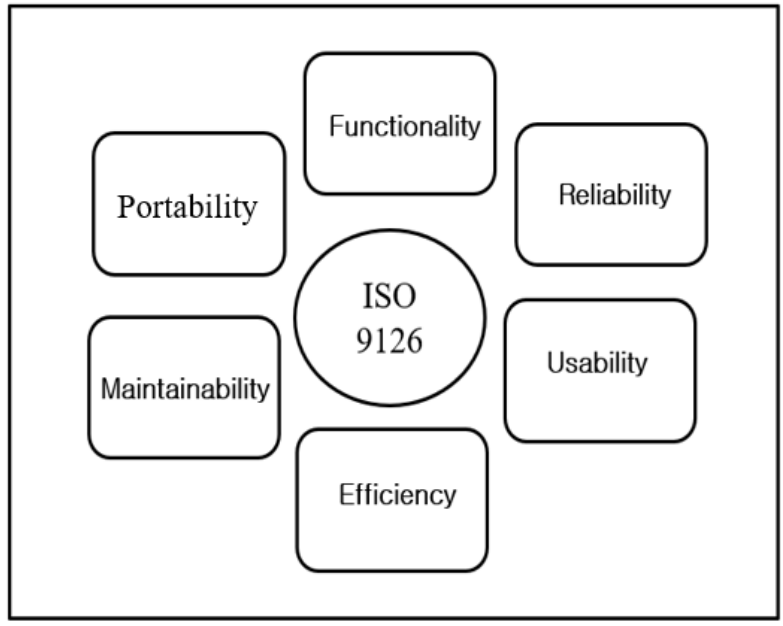


Figure 4. Characteristics of ISO 9126

Functionality:a group of characteristics that affect the presence of a group of functions and the details of their qualities. The activities are those that meet explicit or implicit needs.

Reliability: a group of characteristics that affect whether software can continue to work at a given level under given circumstances for a given amount of time.

Usability: a group of characteristics that affect how much work is required to use something and how each user evaluates it on their own.

Efficiency: a group of characteristics that affect how well the program performs in relation to how many resources is used when certain conditions are met.

Maintainability: a group of characteristics that affect how much work is required to accomplish a given adjustment (which might be anything from software fixes to environmental changes to functional and requirement changes).

Portability:a group of characteristics (including the business, hardware, or software environments) that affect how easily software can be transferred between environments.

2. McCall's Quality Factors

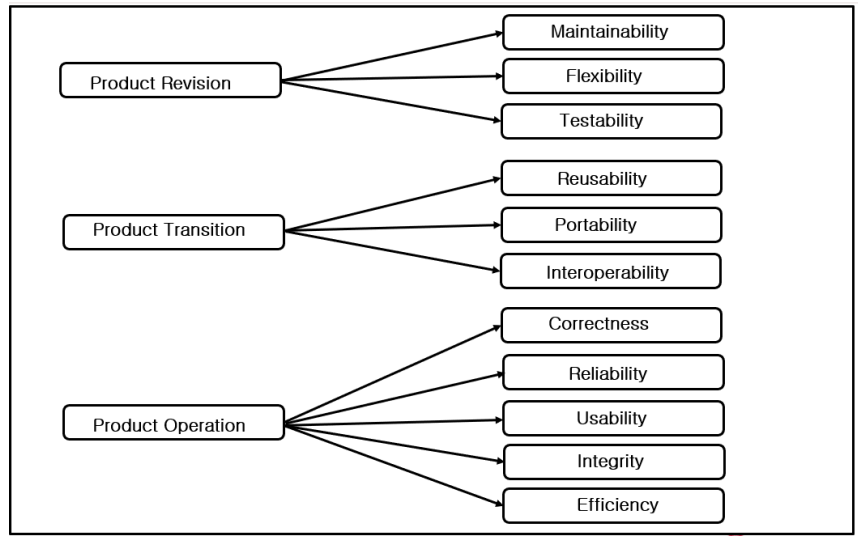


Figure 5. McCall's Quality factors

Correctness: the extent to which a software complies with its requirements and carries out the user's goal objectives.

Reliability: a program's ability to carry out its specified purpose with the necessary accuracy.

Efficiency: computational power and amount of code needed by software to carry out a function.

Integrity: It relates to the security of the software system and preventing access by unauthorized individuals.

Usability: effort needed to understand, use, prepare input, and decipher a program's output.

Maintainability: the time and effort needed to find and correct a flaw in a working software.

Testability: the time and effort needed to check software to make sure it complies with the requirements.

Flexibility: the time and effort required to enhance a working software product.

Portability: effort involved in moving a program between different hardware and/or software environments.

Reusability: Whether or whether a software system's components can be utilized again in other contexts

Interoperability: It means checking whether one software can operate on different platforms.

3. Levels of CMM model:

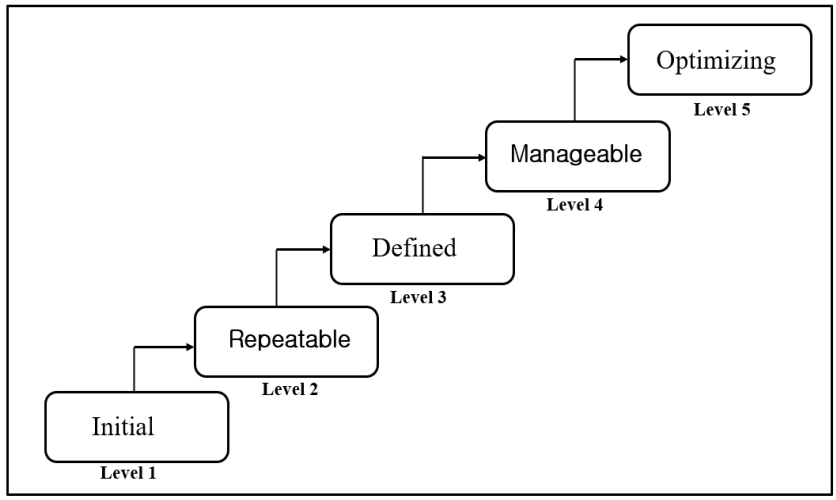


Figure 6. Levels of CMM model

There are five levels in CMM model. They are as follows:

Level 1:the first stages of a new process. Software is created at this level without the use of a processmodels. There is hardly much preparation needed. Even if a plan is made, it might not be carried out. People base their selections on their own qualifications and abilities. At this level, there is a lack of consistency in project management.

Level 2:The process is controlled in accordance with the metrics mentioned in the preceding level at this level. The objectives of software development are repeated.

Studying a setting where good practices are intended to be repeated makes sense.

In the organization, the methods might not be followed again or the same.

To reduce costs and schedule, the company can employ some basic project management techniques.

Level 3: Documentation is crucial at this level. Project management and software development procedures are recorded, examined, standardized, and integrated with organizational processes. In other words, conventional procedures are accepted throughout the entire organization. Software development is done by adhering to a set procedure. Functionalities and the corresponding attributes are monitored.

Cost and schedule are kept in check by monitoring them.

Level 4: Metrics are crucial at this stage. Metrics pertaining to procedures and goods are gathered and examined. These measurements are employed to get quantitative knowledge of both process and product attributes. Corrective measures are initiated when metrics reveal that limits are being breached. For instance, it is helpful to begin a process of root cause analysis if too many test cases fail during system testing in order to comprehend why so many tests are failing.

Level 5: Process management at this level comprises intentional process optimization and enhancement. At this stage, the emphasis is on continuously improving process performance through innovative and gradual technical advancements.

Key practices include Defect prevention, Technology change management, Process change management.

1.4 ERROR

An error results in an incorrect output and generally occurs when there are lot of bugs present inside a program. The words error can be treated as a mistake inside the code. Developers mostly use this word. Incorrect logic, grammar, or loops can result in errors that have an adverse effect on the end-user experience.It arises for several causes, such as application challenges brought on by design, code, or system specification problems.

1.4.1 Types of Errors

Most common types of errors while developing a software are as follows:

1. Syntactic Error:

These errors include mistakes commonly done while writing a code according to the syntax of a language (Java, Python). It can be due to missing parenthesis, error in structure(indentation), or missing operators.

Example 1:

```
int a = 7     //semicolon is missing
```

Error message:

ab.java:20: ‘;’ expected

2. Calculation Error:

These errors are due to bad logic, incorrect conversion from one data to another, incorrect formula, incorrect approximation.

3. Hardware Error:

A hardware error occurs when a computer system's hardware component crashes. Examples:

I/O device errors, unavailable device, incorrect device selection, etc.

4. User Interface Error:

Inability to compile/execute a program as user expects and the reply from the system is very slow. Misleading or confusing information in the help section, unsuitable keyboard usage.

5. Control flow Error:

The errors include datatype error, errors due to exception handling, blocking or unblocking interruptions.

6. Error Handling Error:

Responding to and recovering from erroneous circumstances in your software is the process of error handling.

1.5 TEST CASE

Organizations have several ways of presenting a test case. Nevertheless, a lot of test case templates take the shape of a table. As a tester, creating efficient test cases that thoroughly test a unit is the best approach to ascertain whether the software complies with requirements. The testers can create effective test cases using a variety of test case design methodologies.

When testing, we choose desired preconditions, by giving certain program inputs, and record any observable outputs. When comparing the observed and expected outputs, we determine whether they match, indicating that the test case is successful. If they differ, that indicates a failure condition with the chosen input, and this needs to be carefully recorded in order to identify the problem.

A successful test case has a high likelihood of demonstrating a failure state. Therefore, when creating test cases, test case designers should consider the program's weak points.

1.5.1 Guidelines for designing a Test Case

Every tester should be aware of the following general principles for designing a Test Case.

1. Concentrate on the main purpose of the test case. Include the tests in a concrete manner which should give desirable outputs and cover the whole scenario of the test.
2. Make sure the test case is simple to comprehend and easily understandable by peer testers. Do not make it complex and include what the software is only intended to do.
3. Always create a unique Test ID for each test case so it is easily separable and identified when needed.
4. Include correct test data and recheck the data.

5. Specify the correct expected results.
6. Before continuing, get your test plan reviewed by others.
7. Test cases should be created to handle issues like performance, safety standards, and security requirements as necessary.

1.5.2 A simple Test Case

A test case template offers a versatile yet fundamental structure that you can change as necessary. It can also be used as a checklist to make sure all necessary components are present. Spreadsheets, which have one test per row and the test components in columns, are frequently used by testers.

The following components can be included in the test case:

1. Test Case ID
2. Name of the tester:
3. Test Case description
4. Test Steps
5. Test data
6. Expected Results
7. Actual Results
8. Comments

Date: _____ **System:** _____ **Objective:** _____ **Function:** _____ **Version / Release:** _____ **Status:** _____ (Draft / In Process / Approved)	**Tested by:** _____ **Environment:** _____ **Test ID:** _____ **Req. ID:** _____ **Screen:** _____ **Test Type:** _____ (Unit, Integration, System, Acceptance)

Step Sr.	Step Description	Path & Action	Test Data	Expected Results	Actual Result Pass / Fail	Comments
01						
02						
03						
04						
05						
06						
07						
08						
09						
10						
End						

This test case includes the set of inputs, expected output and actual result. Other information such as name of the tester, environment, test ID, screen is also included in it.

1.6 QUALITY MANAGEMENT (QM)

Quality management is the practice of monitoring the activities, tasks, and processes (inputs) that are utilized to create a product or service (outputs). Quality planning, quality assurance, quality control, and quality improvement are the four primary parts of quality management. Quality Management is the practice of putting all four elements into practice inside a company.

Quality Management places equal emphasis on the tasks and procedures that went into producing the outputs (products and services) as it does on the quality of the outputs themselves. A product's and/or service's quality should ideally improve not only as it is produced, but also as the process itself improves, leading to more reliable, higher-quality goods and services.

1.6.1 Implementing QM

A company must recognize and oversee numerous interconnected, multifunctional activities before implementing a quality management system to help guarantee client satisfaction. The many aims, needs, and goods and services offered by the organization should be taken into consideration when designing the QM. This framework, which is mostly based on the PDCA cycle, enables continual improvement of both the product and the QM. The following are the fundamental actions in putting in place a quality management system:

1. Design
2. Build
3. Deploy
4. Control
5. Measure
6. Review
7. Improve

1.6.2 Importance of QM

The only effective approach to ensure constant, measurable improvement to the firm's operations is by putting in place a comprehensive and standardized quality management system. A quality management system helps with compliance, transparency,and improves customer happiness through documentation and planning.With the aid of the appropriate technology, quality management systems also make sure that opportunities don't go missed.

The benefits of QM are as follows:

1. Improved consistency
2. Increased Profits
3. Continuous Improvement
4. Evidence-based decisions
5. Customer Satisfaction

1.7 SOFTWARE QUALITY ASSURANCE (SQA)

The IEEE Glossary (IEEE, 1991), which is quoted below, provides one of the most widely used definitions of software quality assurance (SQA).

1. A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.
2. A set of activities designed to evaluate the process by which the products are developed or manufactured. Contrast with quality control.

SQA shouldn't be kept to just the development stage. Instead, it should be expanded to include the lengthy years of service that follow the delivery of the product. Incorporating quality concerns that are specifically relevant to the software product expands the concept of SQA by incorporating software maintenance activities.

SQA operations should cover more than just the technical components of the functional requirements; they should also address scheduling and spending issues. This increase in scope is justified by the strong connection between satisfying functional technical requirements and meeting budget or schedule constraints. Professionally "dangerous" adjustments to the project schedule are frequently made when projects are under severe time limitations, which can substantially affect the chances of reaching the functional requirements. Projects that are under financial pressure and unable to manage the insufficient resources assigned to the project and its upkeep might be predicted to have similar unfavorable outcomes.

The new expanded definition of SQA can be given by combining the ISO 9000-3, 1997 with main outlines of the Capacity Maturity Model (CMM). It is as follows:

A systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of a software system product conforms to established functional technical requirements as well as with the managerial requirements of keeping the schedule and operating within the budgetary confines.

The phrase "software quality assurance" may not be entirely accurate. Since "assurance" refers to "grounds for justified confidence that a claim has been or will be achieved," the application of software engineering techniques can only "assure" the quality of a project. In truth, QA is used to lessen the risks associated with creating software that does not satisfy stakeholders' wants, needs, and expectations while staying on schedule and within budget.

SQA might concentrate on software operations, infrastructure, and maintenance in addition to software development. All software processes, from the most fundamental (like governance) to the most complex (like data replication), should be covered by a conventional quality system.

1.8 QUALITY CONTROL (QC)

Quality control refers to the monitoring methods and practices required to meet quality standards.This process entails keeping an eye on and inspecting goods at various points during production or delivery to make sure the intended standard of quality is met. By putting controls in place to regulate and enhance the production or service delivery processes, QC is also concerned with avoiding errors or faults from occurring in the first place.To ensure that products are as uniform as possible and to reduce errors and inconsistencies within them are the two main objectives of quality control.

1.8.1 Key Components of Quality Control

1. Maintaing records:

Keeping meticulous records of inspections, tests, and remedial actions is necessary for maintaining accountability and traceability.

2. Testing:

Executing numerous tests and measurements to evaluate the functioning, performance, or features of goods or services.

3. Inspection:

The frequent examination of goods, materials, or services to spot flaws, violations, or departures from quality standards.

4. Corrective Action:

Putting suitable steps in place to deal with any discovered quality problems and stop them from happening again.

5. Training:

Giving staff the abilities and information required to sustain quality standards successfully.

6. Statistical Process Control (SPC):

You can observe process behaviour, identify problems with internal systems, and resolve production-related problems with the use of SPC tools and methods.

7. Continuous Improvement:

Continually reviewing data and input to spot problem areas and improve the system's overall quality.

1.8.2 Types of Quality Control

Quality control is crucial because it ensures that goods and services are of a high standard, dependable, and satisfy client needs.There are numerous different kinds of quality control techniques, each with a distinct goal and set of procedures. Below is a brief review of the most widely used types of quality control:

1. Process Control:

This kind of quality control focuses on keeping an eye on and controlling the production processes. It is an ability to adjust the process to give desired outputs.To assure quality and boost performance, processes are monitored and modified. To attain consistency, this is often a technical procedure leveraging feedback loops, controls at the industrial level.

2. Control Charts:

For quality control, control charts function as a dashboard. They let you rapidly spot any trends or patterns that might hint at a problem since they demonstrate how a process is doing over time. You can spot problems beforehand and adjust keep your process running smoothly by routinely reviewing control charts.

3. Acceptance Sampling:

Acceptance sampling is like inspecting a random sample of goods from a production batch. This kind of quality control is employed to ascertain whether the batch complies with the necessary requirements. The entire batch is rejected if the sample does not match the standards but is accepted if it does.

4. Product Quality Control:

This kind of quality assurance concentrates on the finished product itself. It entails testing and inspecting the product to make sure it complies with the necessary requirements and standards. Before a product is made available to clients, product quality control helps find any flaws or problems, guaranteeing that they will receive a high-quality product.

1.8.3 Benefits of Quality Control

To guarantee customer satisfaction and establish a solid reputation for a company, quality control is crucial. It can also increase efficiency by streamlining operations and identifying areas for improvement. Costs can be decreased by discovering and fixing flaws early in the production process. Any organization should prioritize quality control because it helps toachieve the following:

1. **Increase Customer Satisfaction:** QC makes ensuring that goods and services meet or beyond consumer expectations, which raises customer happiness and loyalty.

2. **Efficiency:** The production process can be streamlined and made more efficient with the aid of a well-designed and implemented Quality Control method. As a result, productivity rises, waste declines, and profitability rises.

3. **Reduction in Cost:** By finding and repairing errors early in the production process, quality control can assist in cost reduction. As a result, there is less chance of needing to repair faulty items and less need for material waste and rework.

4. **Compliance:** Businesses can lower their risk of facing legal or financial repercussions by putting quality control procedures in place to make sure that their goods or services adhere to the appropriate standards and laws.

5. **Risk Improvement:** QC assists in identifying potential risks and hazards through thorough testing and inspections, allowing organizations to take preventative action.

6. **Constant Improvement:** QC promotes an environment in which businesses constantly work to improve their offerings in terms of goods, services, and procedures.

7. **Innovation:** The identification of chances for innovation and improvement in the production process through quality control procedures results in the creation of fresh, enhanced goods or services. Businesses may stay ahead of their rivals and fulfill the changing needs of their customers by consistently aiming for quality.

8. **Better service quality:** Services can also be subject to quality control procedures to make sure they adhere to the necessary standards. Businesses may enhance the quality of their services and boost customer happiness and loyalty by detecting and resolving issues with service delivery.

1.9 SOFTWARE DEVELOPMENT LIFECYCLE

Most, if not all, technology-based companies have largely adopted the Software Development Life Cycle (SDLC). Learning and comprehending the complexity of the SDLC is becoming more and more crucial due to the

development of technology and the rise in the number of businesses that rely on their own custom apps. There are many famous SDLC models such as Waterfall model, incremental model, Vmodel, iterative model, RAD model, Agile model, Spiral model, Prototype model etc.)

The phases of the software cycle and the order in which they are carried out are described by software life cycle models. Each stage of the life cycle results in deliverables needed by the following stage. Design is translated from requirements. The production of code is done in accordance with the design phase, often known as development. After coding and development, testing compares the implementation phase deliverable to the requirements.

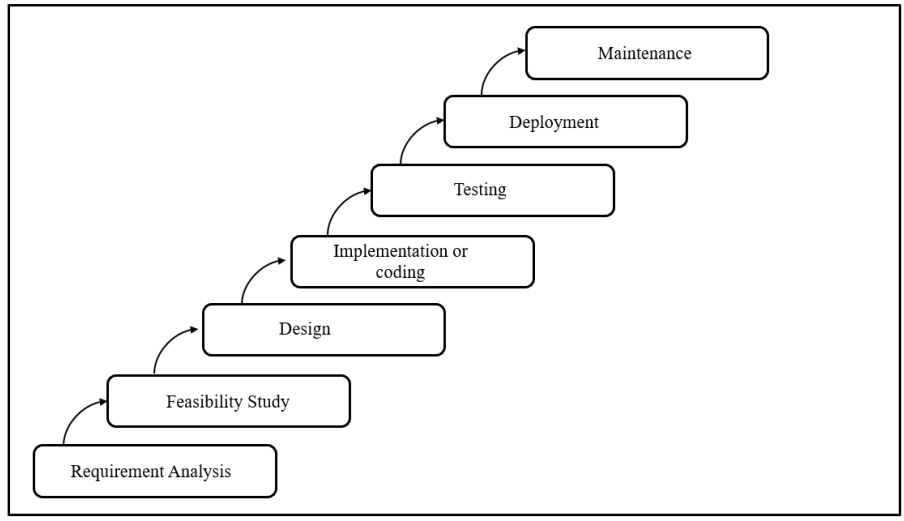


Figure 7. Phases of SDLC

1. Requirement Analysis:

The analysis step involves assembling all the relevant information needed for a new system and coming up with the initial prototype concepts.The project managers and stakeholders are primarily focused on this phase. Developers are free to specify any prototype system requirements, carry out research and analysis to ascertain end-user demands, and analyze alternate solutions to current prototypes. Finally, a Software Requirement Specification document (SRS) is produced as a reference for the model's subsequent phase.

2. Feasibility Study:

If management accepts the system proposal, the next stage is to examine the system's practicality. The primary goal of a feasibility study is to assess the feasibility, user requirements, resource efficiency, and, of course, cost-effectiveness of a proposed system. These are classified into technical, operational, economic, scheduling, and social feasibility categories. The primary goal of a feasibility study is to achieve the scope rather than to solve the problem. During the feasibility study phase, cost and benefit estimations are more precisely made to assess the Return on Investment (ROI). This section lists the resources needed to do a thorough investigation.

3. Design:

The design stage is a prerequisite for the primary developer stage.

Developers will first outline the general application's characteristics, as well as specific components such as its: user interfaces, system interfaces, and network needs. They will usually convert the SRS document they developed into a more logical form that may be later implemented in a programming language. The system's system design can be described using a variety of tools and methodologies: Flowchart, Data flow diagram (DFD), Data dictionary, Structured English, Decision table, and Decision tree are all examples of diagrams.

4. Implementation or coding:

During this phase, programmers actually write code and create the application in accordance with the specifications and design papers from previousstage. The process of turning the program specifications into computer instructions, or programs, is known as the programming phase during this time. Developers will use various tools, including compilers, debuggers, and interpreters, and adhere to any coding standards established by the business. Depending on their requirements, they will select a programming language (C, C++, Java, Python, or C#).

5. Testing:

Software development is not the end. Testing is now required to ensure that there are no defects, and that the end-user experience won't suffer at any time. A test run of the system is completed to remove any bugs before the new system is really put into use. It is a crucial stage in a system's success.

6. Deployment:

The deployment phase starts once the new system has received user approval. After testing, the software's overall design will be completed, and all of the system's programs will be loaded onto the user's computer. The user training process begins when the system has been loaded. After passing this stage, the program is technically ready for market and may be made available to any end users.

7. Maintenance:

Maintenance is required to fix problems that arise with the system while it is in operation and to adjust the system to any changes in the surroundings in which it operates. Developers are also in charge of making any modifications that the software may require after it has been deployed. It has been observed that some flaws are constantly discovered in the systems, which must be noted and fixed. It also refers to periodically reviewing the system. In comparison to smaller systems, larger systems could need more extensive maintenance phases.

1.10 VERIFICATION AND VALIDATION

Verification and validation are two related ideas in software testing that practitioners regularly utilize.

Testing is a combination of both verification and validation.

Software Testing = Verification + Validation

These two are complementary and necessary to one another. Each offers a unique set of error filters and a unique method for identifying software bugs.

1. Verification:

Verification means the process of deciding if the results of a particular phase of the software development process meet the requirements established in the preceding phase.Verification typically requires knowledge of the specific software artifacts, requirements, and specifications. Static testing, which is done manually, relates to verification. We only look over and evaluate the document. Checklists, issue lists, walkthroughs, and inspection meetings can all be used to verify that the product is designed to give all functionality to the client. Verification often comprises reviews and meetings to examine documents, plans, codes, requirements, and specifications.

Definition as per IEEE [IEEE01]:

> "It is the process of evaluating the system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase."

We use verification techniques learned during the early stages of software development, and we check and evaluate the documents produced at the end of each phase.It is therefore the process of reviewing the project's requirement paper, design document, source code, and other associated documents. This manual testing only entails glancing at the documents to make sure the output matches what was anticipated.

2. Validation:

The actual testing of the product takes place after the procedure of verification. This stage involves finding defects that result from differences infunctionality and specifications.

It provides a response to the query, "Are we developing the right product?". The process of figuring out whether the software development process's outputs from a specific phase satisfy the specifications set during the phase before is called validation.

These kinds of activities enable us to verify that a product is suitable for the intended application. Activities aimed at validating a product's

compliance with client expectations. In other words, validation activities concentrate on the finished product, which has undergone rigorous customer testing. Validation determines whether the product lives up to users' overall expectations.

Definition as per IEEE [IEEE01]:

> "It is the process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements."

It needs the software to be run in its entirety. It is dynamic testing, and running the application requires a computer. Here, we encounter failures and pinpoint their root causes.

Effective verification can reduce the requirement for validation and increase the number of faults found in the early stages of software development.

1.10.1 Difference between Verification and Validation:

Verification	Validation
1. The process of document, design, and code verification is static.	1. The process of document, design, and code verification is dynamic.
2. It does not entail running the code.	2. Code is executed
3. It is a manual check of documents/files.	3. Computer based checking.
4. The requirements specifications, application architecture, high-level and detailed design, and database design are the main targets.	4.The ultimate product, as well as individual units, modules, and sets of linked modules, is the target.
5. It employs techniques such as desk checks, walkthroughs, and inspections.	5. It makes use of techniques including white-box, gray-box, and black-box testing.
6. It comes before validation	6. It comes after verification
7. It responds to the query, "Are we building the product correctly?"	7. It responds to the query, "Are we creating the right product?"

1.10.2 SoftwareVerification Methods:

Any type of verification method seeks to identify errors by carefully going over the documents. Numerous techniques, including walkthroughs, inspections, and peer reviews, are frequently employed in practice. Verification aids in preventing potential flaws that could result in program failure.

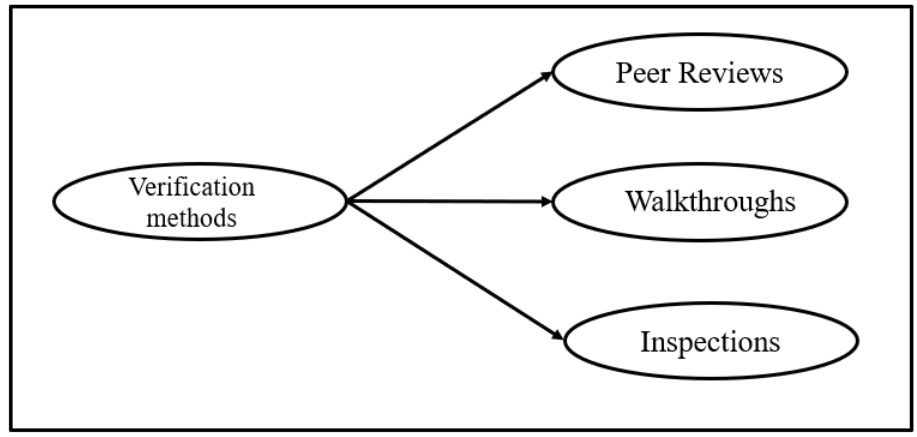


Figure 8. Verification methods

1. Peer Reviews:

The peer-review approach is the simplest and most casual way to examine papers, programs, or software with the aim of identifying flaws throughout the verification process. In this strategy, we hand out the document or software programs to others and ask them to review them. We do this to get their opinions on the quality of our work as well as to get them to point out any flaws. This approach may involve tasks like software verification, SDD verification, and SRS document verification. With this approach, the reviewers may also write a brief report on their findings, observations, etc.Due to the effectiveness and significance of peer review, numerous studies have demonstrated its usefulness. The reviewer has two options for reporting their findings: they can either write a report or just speak up during talks. Peers will engage in this informal activity, which has the potential to be quite productive if reviewers have the necessary topic expertise, programming prowess, and involvement.

Advantages of Peer Reviews:

Without investing a lot of resources, you can anticipate some positive outcomes.

Its structure is significant and quite effective.

Disadvantages of Peer Reviews:

If the reviewer is inexperienced, it could have negative outcomes.

2. Walkthroughs:

Walkthroughs are more formal and systematic than peer reviews. A small group of two to seven people are given a tour of the document by its author. There is no requirement that participants bring anything. The author of the presentation is the only one who prepares for the meeting. All participants have received the document(s). The author introduces the content to familiarize them with it during the meeting. Everyone is welcome to ask questions. For everyone to observe and express their

opinions, each participant may write their observations on any type of display mechanism, including boards, sheets, projection systems, etc. Following the review, the author drafts a report detailing the conclusions and any issues raised during the discussion.

Advantages of Walkthroughs:

It might aid us in identifying potential flaws.

It can also be used to distribute documents to other people.

Disadvantages of Walkthroughs:

The author could omit important details and overly highlight some of his or her interests.

It's possible that the participants won't have many insightful questions to ask.

3. Inspections:

The most formal and organized type of verification approach is an inspection. These are distinct from walkthroughs and peer reviews. The person giving the presentationrather than the document's author—has prepared and read it. This makes them read and understand the text in advance of the meeting. A team of three to six people is assembled, and the team is headed by an objective moderator. Everyone in the group engages honestly, energetically, and in accordance with the guidelines of how such a review should be handled. Everyone may get a chance to voice their opinions, potential flaws, and problematic regions. After the meeting, the moderator's essential suggestions are incorporated into the final report.

Advantages of Inspections:

Finding potential errors or issues in documents like SRS, SDD, etc. may be done extremely effectively using this method.

The critical inspections might also assist in identifying errors and strengthening these documents, which might help in stopping the spread of errors throughout the software development life cycle process.

Disadvantages of Inspections:

They take effort and discipline.

It costs more and calls for qualified testers.

1.11 BLACK BOX TESTING

We are aware that the Testing Technique outlines a method for selecting input test cases for testing and analyzing test outcomes. Structural and functional testing reveals a variety of testing-related aspects. Functional testing or black box testing can be used when the features and operational behavior of the product need to be tested. The benefit of this type of

testing is that the system's internal operations are completely disregarded.

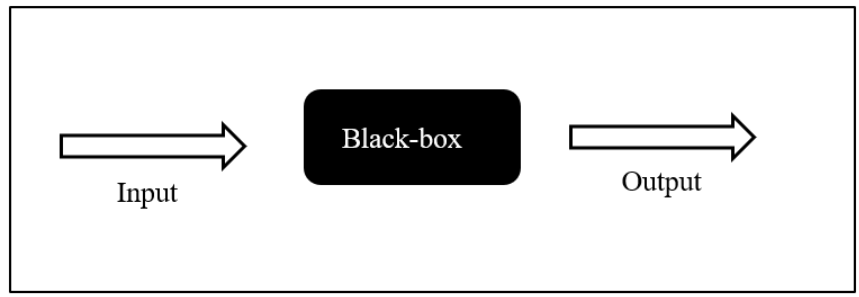


Figure 9. Black-box testing

As you can see in Figure.9, a black-box test excludes the program's internal code and just considers the input and output of the software.The goal of black box testing, also known as functional testing and behavioral testing, is to ascertain whether a software fulfills its functional requirements.During black box testing, creating effective test scenarios is crucial. The tester must entirely rely on the analysis of the transformation of the inputs to the outputs based on which they uncover software faults because they are unaware of how the software functions within. This test enables the tester to determine whether the software performs as intended. Information on the functionalities of the program can be found in the functional specifications or requirements.

Utilizing test monitoring tools is crucial for black box testing strategies.

This is required to keep track of previously run tests, prevent repetition, and help with program maintenance.

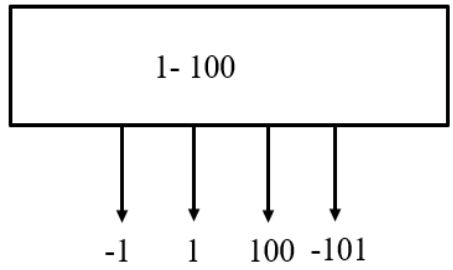
1.11.1 Strategies for Black Box Testing

We want to identify as many flaws as we can in the fewest number of test instances. We employ a few tactics, some of which will be covered in this subsectionto achieve this goal.

1. Boundary Value Analysis:

It is the often employed "black-box" testing, which also serves as the foundation for "equivalence testing." Boundary value analysis uses test scenarios that have extreme values for the test data to evaluate the software. It is used to find the defects or errors brought on by the limitations of the input data. Errors frequently happen at boundary points. Therefore, testing becomes more effective and the likelihood of discovering problems also rises if test cases are created for boundary values of the input domain.

Example: The data should be between 1 to 100, the valid boundary points foe checking will be -1,1,100 and 101.



2. Equivalence partitioning:

We don't want to create multiple test cases that examine the same component of our application in order to reduce testing expenditures. A good test case reveals a new class of faults than previous test cases have.

A technique for minimizing the amount of test cases that must be created is equivalence partitioning. The goal is to divide the input domain of the system into a number of equivalence classes, each of which functions similarly, so that if a test case in one class encounters an error, other test cases in that class would likewise encounter that error.

3. Decision Table Testing:

This method generates test cases based on numerous potential outcomes. In order to pass the test and deliver accurate output, it takes into account a number of test cases in a decision table structure. When there are numerous input options and choices, it is desirable. Decision tables are used to store complex business rules that must be tested before being implemented in a program.Each column in the table represents a particular arrangement of input criteria and is referred to as a rule. Every rule should be turned into a test case.

4. Acceptance Testing:

The customer creates the acceptance test cases. Contracts between the client and the development company for custom software frequently indicate that if the customer's acceptance test cases are not passed, they may refuse to accept delivery of the product.The team occasionally receives the customer's acceptance test cases, which offers them a clear objective to work toward together. Other instances, the client runs the acceptance test cases after obtaining the code but concealing them from the developers.Working openly and together to create the acceptance test cases is something which is far more beneficial for the customer and the development team. The development team and the customer then share a common understanding of what the software must look like for the customer to be satisfied.

5. Graph-Based Testing:

His method creates a connection between logical input known as causes and related actions known as the result. With the use of Boolean graphs, the causes and consequences are displayed. The actions listed below are carried out:

Identify the causes (inputs) and effects (outputs).

Create a cause-and-effect diagram.

Create a decision table from the graph.

Create test cases from decision table rules.

1.11.2 Advantages of Black Box Testing:

The advantages of Black Box testing are as follows:

1. It is simple to write test cases from the viewpoint of an end user, and testers are not required to understand how the software functions within.
2. Black box testing, which is done from the viewpoint of the user, helps to highlight any ambiguities or contradictions in the requirements.
3. The test cases can be created as soon as the product specification is finished.
4. The testers spend little time examining the internal interfaces because they are primarily concerned with the Graphic User Interfaces (GUI) for output. As a result, creating test cases is simple and rapid.

1.11.3 Disadvantages of Black Box Testing:

The disadvantages of Black Box testing are as follows:

1. Bugs may go unnoticed since this form of testing cannot be concentrated on particular functional areas that may be quite complicated.
2. Only a tiny subset of potential inputs can be tested by a tester, and it is very impossible to test every potential input stream.
3. If specifications are not precise and unambiguous, it is quite challenging to create test cases.
4. If the tester is unaware of the test cases the programmer has already tested, things like needless repeating of test inputs may happen.

1.11.4 Key practices while performing Black Box Testing:

This chapter provided several useful pointers for black box testing. The essential elements of effective black box testing are outlined below.

1. Always test frequently and promptly.
2. When creating your test cases, follow the four-item test case template: ID, Description, Expected Results, Actual Results.
3. Instead of testing for what the programmer intended, you should test for what the customer expects the program to perform. It is

recommended to perform black box testing by someone who has a recent, unbiased understanding of the client's requirements.

4. Give specific instructions for how the tester should produce the desired input conditions and how the software should react in the test case.
5. Be clear in this documentation so that other testers besides you can use the test case's instructions to execute the exact same test case. These instructions will be crucial, particularly if a programmer has to recreate a failure.
6. Encourage the consumer to participate in acceptance testing in a collaborative manner.
7. To control the quantity of test cases performed, use equivalence class partitioning. The same flaw will be visible in all test cases belonging to the same equivalence class.
8. Find the very frequent bugs that hide in crevices and gather at boundaries using boundary value analysis.
9. Black box test scenarios only expose the signs of flaws when they uncover failures. You must employ your investigative abilities to identify the flaw in the code that resulted in the failure.
10. Use decision tables to keep track of the intricate business rules that the system must implement and test.

This chapter taught us that comprehensive testing is not practical. However, there are effective software engineering techniques for building test cases that can optimize your likelihood of finding as many flaws as possible with a fair amount of testing, such as equivalence class partitioning and boundary value analysis. It's also advantageous to test the integrated code and integrate code as frequently as feasible. We can isolate bugs in the new code in this way, detect them quickly, and effectively fix them. The advantages of working together with a customer to create the acceptance test cases and automate their execution to create compile-able and executable documentation for the system were the final lesson we learned.

1.12 WHITE BOX TESTING

Glass box testing is another name for white box testing. The tester concentrates on the software code's structure throughout this exam. To verify the software code's logical operation, the tester creates test cases.Software engineers can create test cases that exercise independent paths inside of modules or units, logical decisions on both their true and false sides, loops at their boundaries and within their operational bounds, and internal data structures to test their validity using the white-box testing techniques

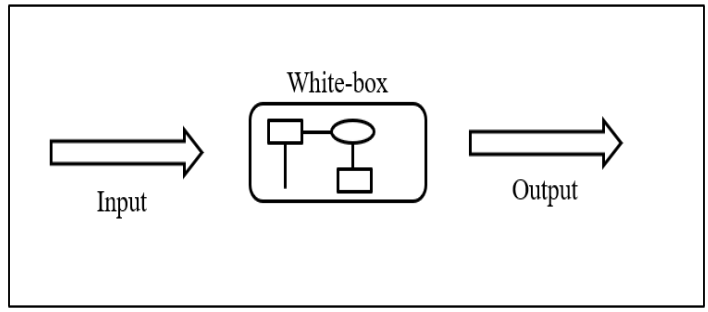


Figure 10. White Box Model

Each software module is individually tested in white box testing. In addition to testing each software module individually, the tester must create test cases that precisely simulate how the modules will interact with one another when the product is run. At the source code level, all tests are executed. The tester examines every aspect of the code, including its effectiveness, branching statements, internal logic, interfaces with external hardware, memory management, code readability, and other factors. Therefore, in order to cover the internal workings of the program, the test cases must be carefully designed.

1.12.1 Strategies for White Box Testing

1. Statement coverage:

The goal of this method is to visit each sentence at least once. Each line of code is therefore tested. Every node in a flowchart needs to be traveled through at least once. It is easier to identify problematic code because every line of code is covered.Using this method, we can determine what the source code should and shouldn't be doing. It can also be used to examine the consistency of the program's various pathways and the quality of the code. This technique's primary flaw is that we are unable to test the false condition in it.

Statement Coverage can be calculated as follows:

$$\text{Statement Coverage} = \frac{\text{No. of statements executed}}{\text{Total No. of statements}} * 100$$

Examples:

Consider the following code. What will be the Statement coverage for the below cases.

Case 1: a = 5, b=2

Case 2: Let a=2, b=5

```
Read a
Read b
if a>b
  print "a greater than b"
else:
  print "b greater than a"
endif
```

Case 1: Let a = 5, b=2

No. of statements executed: 5

Total statements in the code: 7

Statement coverage: 71%

Case 2: Let a=2, b=5

No. of statements executed: 6

Total statements in the code: 7

Statement coverage: 85.20%

2. Branch Coverage:

This method involves creating test cases that traverse every branch from every decision point at least once. Every edge in a flowchart needs to be traveled along at least once.

In order to make sure that the program is reliable and that all potential routes through the application have been adequately tested, branch coverage testing is crucial. We will examine branch coverage testing in more detail in this post, including what it is, how it operates, and its significance.

Branch coverage can be calculated as follows:

$$\text{Branch Coverage} = \frac{\text{No. of branches executed}}{\text{Total No. of branches}}$$

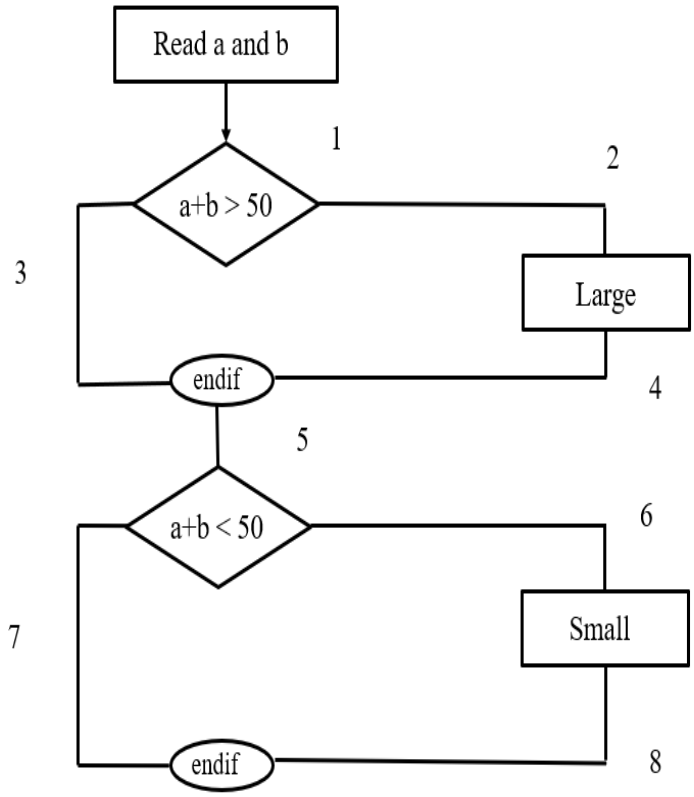
Examples:

Consider the following code. What will be the Branch coverage?

```
Read a
Read b
if a+b > 50 then
  print "Large"
endif
if a+b < 50 then
  print "Small"
endif
```

Let's draw a flowchart.

Consider the following code. What will be the Statement coverage for the below cases.



The smallest number of pathways necessary to cover all the edges must be identified in order to determine branch coverage.

Case 1: traversing through "Yes "decision, the path is 1-2-4-5-6-8 but edges 3 and 7 are not covered in this path.

Case 2: To cover the edges 3 & 7 as these are not covered in the first step, we have to traverse through "No" decision. In the case of "No" decision the path is 1-3-5-7.

So by traveling through these two paths, all branches have been covered.

Branch Coverage is 2.

3. Condition Coverage:

Condition coverage is a proportion of the program's Boolean sub-expressions that have been evaluated in test cases as having either a true or false result. The results of each of these sub-expressions are measured separately by condition coverage. You may be sure that each of these subexpressions has been separately verified as true and false by using condition coverage.

1.12.2 Advantages of White Box Testing:

The advantages of white box testing are as follows:

1. Code lines that are excessive or unnecessary and may cause undiscovered bugs can be cut out.
2. Testing is done at the code level, hence it aids in code optimization.
3. Since the tester is familiar with internal coding, creating test cases to adequately test the product is fairly simple.

1.12.3 Disadvantages of White Box Testing:

The disadvantages of white box testing are as follows:

1. It is virtually impossible to inspect every line of code for hidden faults or errors that could cause the software to malfunction.
2. This test takes a long time to complete for applications that are complicated.
3. Because skilled testers are needed to do this test, the expense goes up.

1.12.4 Key practices while performing Black Box Testing:

This chapter provided several useful pointers for white box testing. The essential elements of effective black box testing are outlined below.

1. Write enough white box test cases to at the very least cover all of your statements.
2. Get the highest level of decision/branch and condition coverage possible.
3. Draw the flow diagram for a section of code until you are more comfortable calculating cyclomatic complexity.
4. The study of control flow-based unit testing should be done using an automated coverage monitor.

1.13 COMPARISON OF BLACK BOX TESTING WITH WHITE BOX TESTING

Black Box Testing	White Box Testing
1. It is also called as Functional testing	1. It is also called as Structural Testing
2. It uncovers different classes of errors.	2. It mainly focuses on errors related to internal logic
3. It is mostly applied in later stages of testing	3. It is mostly applied in earlier stages of testing
4. Control structure of a program is not considered.	4. Control structure of a program is considered.
5. It is called as "testing in the large".	5. It is called as "testing in the small".
6. It consists of the testing carried out at the software interface.	6. The intricacies of the procedure are carefully examined.
7. It finds errors such as performance errors, interface errors, incorrect or missing functions	7. It finds errors related to internal logic and status of the program.

1.14 LET US SUM UP

We discussed software testing and quality management in this chapter. All software engineers today need to be familiar with the principles and practices of software testing. The ideas in this chapter will likely be applied frequently while testing software. The activities of a test engineer are described in this chapter, along with several critical phases of a SDLC (Software Development Lifecycle). The two steps of verification and validation provide more detailed insights into testing process. The fundamental concept of black box and white box testing is also explained in the later part of this unit. Testing takes up roughly 70% of development time. In this chapter, we examine this, and many more intriguing concepts related to testing and quality.

1.15 LIST OF REFERENCES

1. Myers, G.J., Badgett, T., Thomas, T.M. and Sandler, C., 2004. The art of software testing (Vol. 2). Chichester: John Wiley & Sons.
2. Kaur, Manpreet. "Software Testing and Quality Assurance." (2012).
3. Ammann, Paul, and Jeff Offutt. Introduction to software testing. Cambridge University Press, 2016.
4. Jorgensen, Paul C. Software testing: a craftsman's approach. Auerbach Publications, 2013.

1.16 BIBLIOGRAPHY

1. Naik, Kshirasagar, and Priyadarshi Tripathy. Software testing and quality assurance: theory and practice. John Wiley & Sons, 2011.
2. Froberg, Scott. "Software Testing by Yogesh Singh." ACM SIGSOFT Software Engineering Notes 37.3 (2012): 36-36.
3. Mauch, Peter D. Quality management: theory and application. CRC press, 2009.
4. William (Informático) Perry. Effective methods for software testing. John Wiley & Sons, 1995.
5. Chopra, Rajiv. Software testing: a self-teaching introduction. Mercury Learning and Information, 2018.
6. Chopra, Rajiv. Software testing: a self-teaching introduction. Mercury Learning and Information, 2018.

1.17 UNIT END EXERCISES

1. What are the qualities of a good tester?
2. Give a brief history on Quality Revolution.
3. Explain the 5 levels of CMM model.
4. Explain Shewhart cycle.
5. What are the guidelines for designing a test case?
6. Explain SDLC.
7. Briefly explain strategies of white box testing.
8. Calculate the statement coverage if the number of statements executedis 10 and total statements in the code are 15.
9. State the advantages and disadvantages of black box testing.
10. Compare black box testing and white box testing.

2

SOFTWARE QUALITY ASSURANCE

Unit Structure :

2.0 OBJECTIVES

After completion of this module, you will learn:

- What is Software Quality Assurance and why it is needed?
- What procedures, techniques, and activities are involved in software quality assurance?
- The current SQA practices and standards.
- Pareto diagrams, cause-effect diagrams, scatter diagrams, run charts.
- Quality costs, types of quality costs

2.1 INTRODUCTION

In this unit, we are going to study Software Quality Assurance (SQA). Software quality assurance (SQA) is a tough accomplishment. While standards clarify how to maximize performance, Quality Assurance Engineers are largely allowed to make practical decisions about how to achieve the SQA. The purpose of this study is to understand SQA's importance in delivery of a Software.

Individuals create, maintain, and use software in a wide range of circumstances. Software is created by students in their classrooms, enthusiasts join open-source development teams, and professionals produce software for a variety of industries, from aerospace to banking. Each of these distinct groups will need to address quality issues that develop in the software they are using. This chapter will state the SQA terms, highlight the source of errors in software and discuss software engineering practices to be followed depending on the kind of working sector.

Each profession has a set of guiding principles which are to be followed by the professional. To be a professional individuals should be aware of these principles or have work experience for the same.To achieve Quality Assurance for a software, individual should have a good understanding of,

1. Software quality fundamentals
2. Software Quality Management Processes
3. Practical considerations
4. Software Quality tools

2.2 QUALITY CONCEPTS

Software quality does not have a single, all-encompassing definition. This is due to the complexity brought on by the three or more parties – the customer, developer, and stakeholders – who are impacted by the quality of software.

Some people think software quality evaluations should focus on client happiness, but it is not necessary that the quality that the client wants meets the other standards.

2.2.1 What is Software Quality?

Let's say that either individually or as a group, we create a product. Because this product is being created to address a problem that exists in nature, there are consumers that are interested in buying our product as they are having similar issues. Our product is fixing the issue; however, it is enough to sell the product to the user. We must guarantee the quality of your product to the consumers. We do that by informing the consumer about tests we performed against the factors that assure us the product being fault free.

Similarly, when software is built to solve a problem, we need to assure its quality to end-users. Software that solves the end-user's problem but ends up giving error or vulnerable to attackers or crashing the user's system, etc. We call it bad quality software. Therefore, good quality software is tested against the SQA standards to ensure its quality. The standards are declared by ISO (International Organization for Standardization) and other organizations which certify that our software follows the standard by clearing the tests and it is good quality software. These are the officially certified software the consumers prefer to use as they make the least compromise to their systems as compared to the non-certified ones.

The software quality is defined as healthy software which gives almost no error, least vulnerable to attackers, tested and certified under the SQA standards and solves the problem efficiently with maximum accuracy.

To sustain the software quality, the software should not cause incidents like,

1. The system crashed during production.
2. The developer made an error.

3. We examined the test plan and discovered a flaw.
4. We discovered a bug in a program.
5. The system failed.
6. The client expressed concern over a computation error in the payment report.
7. The monitoring subsystem was said to be experiencing a problem.

2.2.2 Defect in Software

Software defect that degrades the software quality are caused by different factors. They can be causeddue to ignored use cases or exceptional cases occurring.

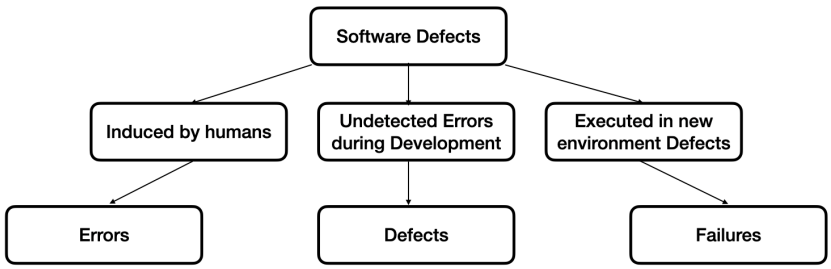


Figure 1. Software Defects

Figure 1 shows the software defects caused by humans, errors during development, and new environment defects. Firstly, the software defects are inserted by humans which we state as the development errors which can be observed and fixed before delivering to the consumer by following SQA standard tests. Secondly, the defects are undetected before deployment as they are caused by ignorance testing through different use cases. Lastly, the defects caused when running the software in a different client environment. Example: if the client tries running the software with low RAM the system crashes. Therefore, different environments have different factors that cause software failure. However, the Errors can be tackled before deployment, defects can be fixed after testing it on production environment and the failures are exceptional factors occurring can be solved with next software upgrade.

Every stage of the software development life cycle has the potential to introduce errors, including those in the requirements, code, documentation, data, and tests.

Human error on the part of users, clients, analysts, designers, software developers, testers, or testers is nearly always the root of the problem. Everybody involved in the software engineering process will need to be able to use a classification of the sources of software error by category that is created by SQA.

Here are eight common error-cause categories as an illustration:

1) issues with defining requirements
2) maintaining effective client-developer communication
3) deviations from specifications
4) errors in architecture and design
5) errors in coding (including test code)
6) non-compliance with current processes/procedures
7) insufficient reviews and tests
8) errors in documentation.

2.2.3 Software Quality Views

Development and purchase of software products are influenced by a variety of factors.These considerations include user wants and expectations, manufacturer considerations, a product's intrinsic qualities, and perceived worth. It's crucial to look at quality from a wider angle to fully understand it. This is since the idea of quality predates the creation of software.

Figure.2 displays the Software Quality Views. They are as follows:

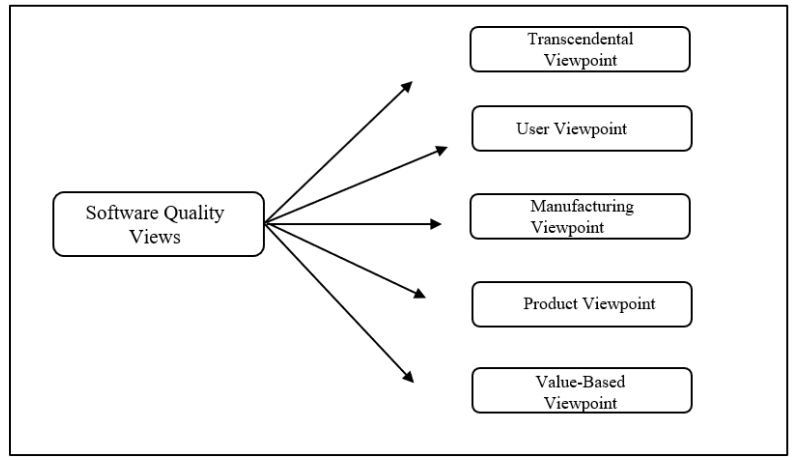


Figure 2. Software Quality Views

1. **Transcendental Viewpoint:** According to the transcendental perspective, quality is something that can be identified via personal experience but is not formally defined. It is believed that quality is an ideal that cannot be clearly defined since it is an ideal that is too complex. A high-quality item, however, is noticeable and distinct. No attempt is made to convey the transcendental view using concrete examples due to its philosophical nature.
2. **User Viewpoint:** The degree to which a product satisfies a user's needs and expectations is considered by the user perspective. The service clauses in the sales contract have an impact on quality in addition to how well a product will perform. In this viewpoint, a user is worried about a product's suitability for use. This view has a unique character. A product is deemed to be of good quality if it meets the

needs of a sizable number of clients due to the personalized nature of the product's view. Finding out which product features customers value highly is useful.

3. **Manufacturing Viewpoint:** The manufacturing view originated in manufacturing-related industries, such as the electronics and automotive industries. This perspective sees quality as meeting standards. Any deviance from the established standards is considered as lowering the product's quality. The manufacturing perspective places a lot of importance on the idea of process. Manufacturing should be done "right the first time" to trim down on development and maintenance costs. Nevertheless, there is no assurance that adhering to process norms would result in high-quality products. Some challenge this theory with the claim that conforming to a process can only result in uniformity in the product and that it is therefore possible to create poor products consistently. But product quality can be recursively enhanced.

4. **Product Viewpoint:** If a product is made with good internal properties, it will also have good outward attributes, according to the main principle of the product view. The product view is appealing because it creates a chance to investigate causal linkages between a product's internal attributes and exterior qualities. According to this perspective, a product's current quality level reveals whether it has measurable product attributes. It is possible to evaluate product quality objectively.

5. **Value-Based Viewpoint:** The value-based perspective represents a combination of two concepts: excellence and worth. Quality is a state's excellence, and value states its worth. The central idea in the value-based view is how much a consumer will pay for a certain level of quality. The reality is that quality is meaningless if the product does not give out financial profits to the organization. Essentially, the value-based view represents a trade-off between finance and quality.

2.2.4 Software Quality Requirements

The software quality can be assured, if it follows software requirements management, elicitation,analysis, specification, and validation. Figure 3 states the types of requirements a software should satisfy.

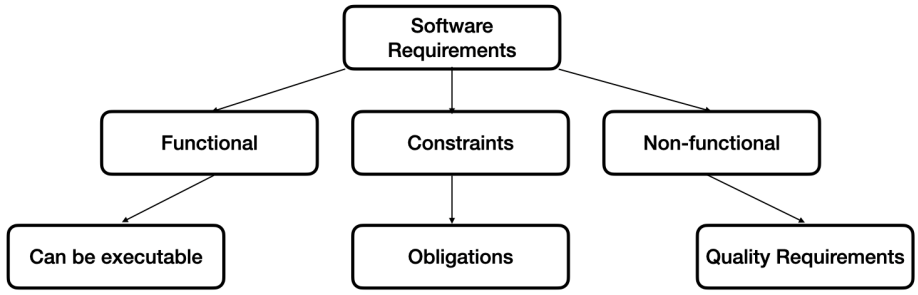


Figure 3. Software Requirements

The Functional requirement is a possible use case for software to achieve. The constraint requirements are the obligations the software needs to prove wrong. And the Non-functional requirements quality are the requirements from which we can prioritize the important ones and rest can be satisfied in later versions.The requirements should be properly documented and communicated to all shareholders of the software.

A good quality software requirement satisfies the following criteria:

1. It should be correct, complete, and clear for each stakeholder group, including the client, the system architect, testers, and those responsible for system maintenance
2. it is said to be of high quality.
3. clear, meaning that all parties involved are giving the same interpretation of the requirement
4. It should be precise, consistent, and feasible to achieve.
5. independent of the design
6. independent of the implementation technique
7. verifiable and testable
8. may be traced back to a business need
9. unique.

2.3 SQA ACTIVITIES

A Quality Management System (QMS), which is made up of numerous components and is a part of the larger system of software development that includes project, process, and product management systems, ensures SQA.

The Software Engineering Institute (SEI) suggests a series of actions that, when successfully carried out, guarantee the quality of the design. These actions consist of:

1. Quality assurance planning
2. Data gathering on key quality defining parameters.
3. Data analysis and reporting
4. Quality control mechanisms

The most important criterion for SQA is that it be a separate group in charge of quality within the company. They establish the objectives, benchmarks, and systems (mechanisms) for SQA. The SQA team's responsibility is to help the software development team manage the software's quality requirements. Every piece of software has quality

objectives that the customer has set. The development team must meet these quality objectives by implementing a series of procedures or making sure that the consumer receives quality.

SQA operations follow the standard procedures of quality management. These tasks serve as monitoring, tracking, evaluations, audits, and reviews to make sure the organization's quality policy is followed. Independently completing these tasks and providing the development team with feedback.The development team oversees providing the customer with the necessary quality. The development team is responsible for implementing quality policy in terms of goals, objectives, practices, checks, controls, documentation, and management input. For instance, the quality policy mandates the creation of a test plan for both the early stages of development and the final stages. SQA can apply this policy using several tools such as auditing and inspection.

Verify adherence to the standards and procedures outlined in the QA policy; discrepancies are corrected. Ascertain that deviations are recorded, documented, and entered the QA database for reference. To make sure that standards are met, and consumer quality is guaranteed, design and architecture are examined. Activate change management. Gather information on numerous observations made during the auditing, inspection, and review processes to create a QA database and improve various standards.

2.4 SOFTWARE REVIEWS

A process or meeting during which a software product is examined by a project staff, managers, users, customers, user representatives, or other interested parties for comment or approval," according to the definition of a software review.

The term "software product" here refers to "any technical document or partial document, produced as a deliverable of a software development activity" and can refer to contracts, project plans and budgets, requirements documents, specifications, designs, source code, user documentation, support and maintenance documentation, test plans, test specifications, standards, and any other kind of specialized work product.

Reviews of software can be categorized into three groups:

- Software peer reviews are conducted by one or more engineers, to evaluate the technical content and/or quality of the work.
- Software management reviews are carried out by management representatives to assess the state of the job completed and to make choices about subsequent actions.
- Software audit reviews are carried out by individuals who are not involved with the software project, and they assess how well the project complies with the requirements, standards, contracts, and other criteria.

- Peer Review Types:
- Code review is systematic analysis of computer source code, frequently done as peer review.
- Pair programming is a form of code review in which two developers work on the same piece of code at the same time.
- Inspection is a peer review that is conducted in a very rigorous manner, with each reviewer using a specific process to look for errors.
- Walkthrough is a author guides members of the development team and other interested parties through a software product in a sort of peer review in which the participants comment on flaws and offer questions.
- Technical review is a form of peer review in which a group of knowledgeable individuals assesses the software product's ability for its intended purpose and identifies deviations from requirements and standards.

2.4.1 Formal Technical Reviews

A software quality assurance task carried out by software engineers is formal technical review (FTR).

FTR's formal technical review (FTR) goals are: These include:

- For any representation of the software, it might be useful to find errors in logic, function, and implementation.
- FTR checks that the software satisfies predetermined requirements.
- To make certain that software is represented in accordance with established standards.
- Reviewing the consistency of software that is being developed in a uniform way is helpful.
- To improve project management.

Additionally, the goal of FTR is to give junior engineers a better opportunity to watch the analysis, design, coding, and testing process. Additionally, FTR promotes backup and continuity so that users can become familiar with software components they might not have otherwise encountered. In reality, FTR is a class of reviews that also includes small-group technical evaluations of software during walkthroughs, inspections, and round-robin reviews. Each FTR is run as a meeting, and it can only be called successful if it is well-organized, managed, and attended.

Everyone who attended FTR must make a decision after the review.

Accept the product exactly as it is.

Reject the project because of a significant error (after it has been fixed, a different app has to be approved), or

Accept the product provisionally (there are some minor problems that should be fixed, but no more review is necessary).

2.4.2 Peer reviews:

A team of three to five people works best for peer reviews. The addition of one to three more participants is permitted in some circumstances.

All of the attendees ought to be the software system creator and author's peers.A recommended peer review team includes:

- A review leader
- The author
- Specialized professionals.

1. The review leader:

The role of review leader ("moderator" in inspections, "coordinator' in walkthroughs) differs only slightly by peer review type. Candidates for this position must:

- Be knowledgeable about how projects of this type are developed, as well as their technologies. It is not necessary to have prior knowledge of the project under consideration.
- come from a different project team.
- Keep in touch with the author and the development team in a positive manner.
- Show that you have experience planning and running business meetings.

2. The author:

Each method of peer review requires participation from the author.

3. Specialized professional:

The specialized professionals participating in the two peer review methods differ by review. For inspections, the recommended professionals are:

- **Designer:** A designer is the systems analyst who oversaw the software system under review's study and design.

- **Coder:** A professional who is well-versed in coding activities, preferably the team leader for the specified coding team, is referred to as a programmer or implementer.
- **Tester:** A tester is a skilled professional—ideally the team leader—who concentrates on identifying design flaws that are typically found during the testing stage.

2.5 STATISTICAL QUALITY ASSURANCE

Software quality can be attained through competent analysis, design, coding, and testing, as well as by using formal technical reviews, a testing strategy, better control of software work products and the changes made to them, and the use of accepted software engineering standards. Additionally, a wide range of quality criteria can be used to define quality, and quality can be (indirectly) quantified using a number of different indices and metrics.

Every programming language has a syntax and semantics that can be defined, and attempts are being made to create a similarly rigorous method for describing software requirements. Applying mathematical proof of correctness to show that a program responds perfectly to its specifications should be possible if the requirements model (specification) and the programming language can be described in a rigorous way.

1. The first step in statistical SQA is the collection and classification of data on software problems.
2. Attempts are made to identify the root cause of each issue, such as non-conformance with specifications, a design flaw, a standard violation, or poor customer communication.
3. Isolate the 20% (the "vital few") using the Pareto principle, which states that 80% of faults may be attributed to 20% of all potential causes.
4. Once the essential few reasons have been found, address the issues that led to the faults.

An important step towards the development of an adaptive software engineering process, in which changes are made to strengthen the process's error-introducing components, is represented by this very straightforward idea.

Software statistical quality assurance strategies have been found to significantly increase quality. In certain instances, software organisations have used these strategies to reduce faults by 50% annually.

2.6 SOFTWARE RELIABILITY

Reliability is a very broad notion that can be used whenever someone anticipates something or someone else to "behave" in a specific manner. One of the measures used to assess the quality of a software system is reliability. It is conceivably the most crucial aspect of a product's quality to consider. In terms of system operation, reliability is a user-focused quality factor that considers how frequently systems fail. Intuitively, a system is thought to be more reliable than one that fails more frequently if users only sometimes experience system failure.

The perception of a system's reliability decreases when they notice more and more system failures. In an ideal world, users of software systems would never experience a system failure, deeming the system to be extremely reliable. Considering that real-life systems are intrinsically complicated, creating a "correct" system—that is, a fault-free system—is a challenging endeavourin and of itself. When real-world conditions are taken into account, the task of developing a fault-free system becomes increasingly challenging.

In the case of software system development, for instance, businesses may not always have the resources necessary to create a highly reliable system, even in the best-case scenario when they have a team of highly qualified and experienced employees. A corporation may not be able to make an effort to create a "correct" system if they use the market window concept. A market window is seen as the window of opportunity for the launch of a product before it is surpassed in terms of capabilities or price by a different offering from a rival vendor. Companies may compromise on reliability in order to reduce costs and achieve delivery deadlines due to economic factors.

Users can tolerate certain failures and there are an unknown amount of defects in a delivered system, hence it is ideal to describe system reliability as a continuous variable rather than a Boolean variable. Higher levels of reliability are typically the result of more work put into the development process. Conversely, less work results in less reliable systems. As a result, reliability can be used to determine the significance of trends, define objectives, and forecast when those objectives will be met. Developers might be curious about how a particular development method, the duration of system testing, or a design review technique affects software reliability, for instance. Developers may be curious about the frequency of system failure in a particular operational environment.

Software maintenance entails modifying the system in a variety of ways, including by changing the requirements, the design, the source code, and the test cases. The software experiences a period of instability while performing those modifications. The system's decreased reliability is the result of the instability. When a system is maintained, its reliability level declines since new problems could be created as a result of all those modifications. It makes intuitive sense that less changes made to a

system would result in less degradation of its present reliability level. On the other hand, if too many changes are made to the system at once, the level of reliability may be dramatically reduced. As a result, the degree of change how much reliability one is willing to give up for the time being determines what changes should be made to a product at a given time.

2.7 THE ISO 9000 QUALITY STANDARDS

The ISO has created a number of standards known as the ISO 9000 collectively. With its headquarters in Geneva, Switzerland, the ISO was established in 1946. In the areas of quality assurance and quality management, it creates and advances worldwide standards. The ISO 9000 standards are typically applicable to all tangible goods produced by human labour, such as spices and software. Some brands of rice and spices that are used in everyday cooking even claim to be ISO 9000 certified. Every 5-8 years, or so, the ISO 9000 standards are reviewed and updated. The ISO 9000:2000 designation refers to the most recent ISO 9000 standards, which were published in 2000. The ISO 9000:2000 standard has three parts, which are as follows:

ISO 9000 : Fundamentals and vocabulary

ISO 9001 : Requirements

SO 9004 : Guidelines for performance improvements

Now that ISO 9002 and ISO 9003 are no longer included in ISO 9000:2000, we would like to remind the reader that they were once a part of ISO 9000:1994. The quality system model for quality assurance in manufacturing and installation was covered by ISO 9002, whilst final inspection and testing were covered by ISO 9003.

2.7.1 Principles of ISO 9000

1. Customer satisfaction is the key to an organization's success. An organization needs to consistently comprehend its clients' wants. Understanding the needs of the consumer is essential for doing so. The mere satisfaction of client demands is insufficient. Instead, businesses must strive to exceed client expectations. One can better grasp consumers' implicit expectations and unmet requirements by getting to know them. It is important for individuals working in many departments of an organization, such as marketing, software development, testing, and customer service, to have a common understanding of the consumers and their needs.

2. **Leadership:** Leaders determine the course that their organisation should take, and they are responsible for clearly communicating this course of action to every party involved. A cohesive understanding of the organisational direction is necessary for every individual inside a company. Employees will struggle to know where they are going if

they do not have a clear sense of the organisational direction. Setting ambitious yet doable goals and objectives is a leadership requirement. Leadership should recognize employee contributions. Leaders foster a supportive environment so that the team members can work together to achieve the organization's objective.

3. **Participation of People:** Organisations depend on people in general. People are involved in decision-making at all levels and are informed of the organisational orientation. People are given the chance to hone their strength and put their skills to work. People are urged to use creativity when carrying out their duties.

4. **Process Approach:** The concept of process can be used to complete important activities in a number of ways that are advantageous. A process is a set of steps that converts inputs into outputs. By making the process clear, repeatable, and quantifiable, organisations may create a plan that includes allocating resources and scheduling activities. As a result, the organisation improves in efficiency and effectiveness. Processes that are continually improved are more effective and efficient.

5. **Management with a systemic approach:** A system is a collection of interconnected processes. A system of interconnected processes can be used to conceptualise a whole organisation. In the context of software development, there are numerous processes that can be named. For instance, obtaining customer requirements is a unique process requiring specialised knowledge. Another unique technique is creating a functional specification using the requirements as input. In an organisation, processes are run sequentially and concurrently. People are constantly engaged in one or more processes.

6. **Continuous Improvement:** Continuous improvement refers to the periodic reviews of the processes involved in creating, for example, software products to determine where and how they might be further improved. Since no process can ever be perfect from the start, continuous improvement is crucial to an organization's success. It is normal to examine the procedures and look for improvements given the independent changes occurring in many sectors, such as customer perceptions and technologies. Lower maintenance and manufacturing costs are the outcome of ongoing process advancements. Additionally, ongoing advancements result in less discrepancies between expected and actual behaviour

   Organisations must establish their own guidelines for when to launch a process review and specify the review's objectives.

7. **Factual Approach to Decision Making:** Facts, experience, and intuition can all be used to inform decisions. Through the use of a reliable measurement technique, facts can be acquired. The core of measurement is the identification and quantification of parameters. It is simpler to establish techniques to measure elements once they have been quantified. Methods are required to verify the measured data and

make the data accessible to those who require it. The measured data must to be precise and trustworthy. Organisations can determine the extent of process improvement by using a quantitative measurement programme.

8. **Relationships with suppliers that are mutually advantageous:** Companies rarely manufacture all the parts that go into their products. Organisations frequently purchase parts and supporting systems from outside vendors. The providers must be carefully chosen by the organisation, and needs and expectations must be communicated to them. The performance of the externally purchased goods should be assessed, and the suppliers should be informed of the need to enhance their goods and procedures. It is important to have a cooperative, mutually beneficial relationship with the suppliers.

2.7.2. Advantages of ISO 9000:

It is generally accepted that effective quality management enhances operations, frequently favorably affecting investment, market share, sales growth, sales margins, competitive advantage, and litigation avoidance. Wade and Barnes agree that the quality standards in ISO 9000: 2000 are sound and that the standards "provide a comprehensive model for quality management systems that can make any company competitive." Barnes also cites studies by Lloyd's Register Quality Assurance and Deloitte Touche that found ISO 9000 boosted net profit and that the expenses of registration were recouped in three years, respectively. Implementing ISO frequently results in the following benefits, according to the Providence Business News:

1. Create a more efficient , effective operation
2. Increase customer satisfaction and retention
3. Reduce audits
4. Enhance marketing
5. Improve employee motivation, awareness, and morale
6. Promote international trade
7. Increase profit
8. Reduce waste and increases productivity

However, a large statistical analysis of 800 Spanish companies indicated that simply registering for ISO 9000 makes little difference because most interested organizations had already committed to quality management in some way and were operating at par before registration.

More and more businesses are embracing ISO 9000 as a business tool in today's economy, which is dominated by the service sector. Companies are employing IO 9000 procedures to boost their productivity and profitability through the use of clearly specified quality objectives, customer satisfaction surveys, and a well-defined continuous improvement program.

2.8 SIX SIGMA

One of the most well-liked quality techniques currently is Six Sigma. With only 3.4 faults per million units or operations (DPMO), it is the

mark that denotes "best in class." When properly used, this approach yields impressive and noticeable increases in quality. Six Sigma methods are currently being used in a huge range of organizations and in a huge range of roles.

The technique is proven to be more than just a quality program, as seen by its effectiveness at major corporations including Motorola, General Electric, Sony, and Allied Signal. Why is Six Sigma being adopted by these big businesses? What distinguishes this methodology from others?

The objective of Six Sigma is to increase profitability, not to reach six sigma levels of quality. Prior to Six Sigma, enhancements made possible by quality initiatives like Total Quality Management (TOM) and ISO 9000 frequently had no discernible effect on a company's net profits. In general, these high-quality programs came into being gradually as a result of unseen influence and intangible progress.

Six Sigma was created as a collection of procedures to enhance manufacturing procedures and get rid of flaws, but its use was later expanded to other kinds of corporate procedures as well. Any issue that results in customer unhappiness is referred to as a defect in Six Sigma.

- Six Sigma stands for six standard deviation from mean (sigma is thee Greek letter used to represent standard deviation in statistics).
- Six Sigma methodologies provide the techniques and tools to improve the capability and reduce the defects in any process.
- Six Sigma strives for perfection. It allows for only 3.4 defects per million opportunities (or 99.999666 percent accuracy)
- Six Sigma maintains consistent output quality while reducing variation and enhancing process performance. As a result, there are fewer defects, higher revenues, higher-quality products, and happier customers.
- The fundamental concepts and methods used in business, statistics, and engineering are all incorporated into Six Sigma.
- The Six Sigma principle's goal is to produce goods and processes with zero faults.
- The Six Sigma principle's goal is to produce goods and processes with zero faults.
- It permits 4.4 mistakes for every million possibilities.

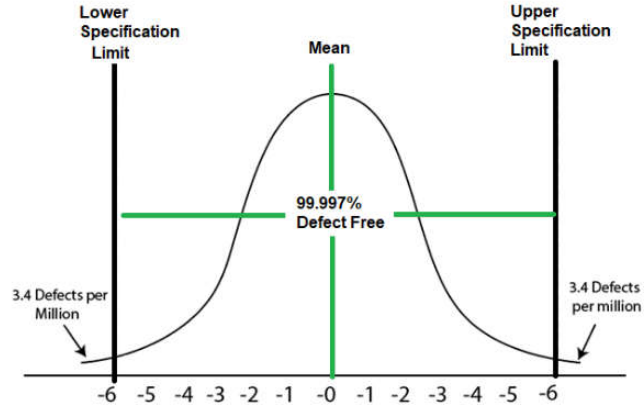


Figure 4. Six Sigma Curve

2.8.1 Characteristics if Six Sigma:

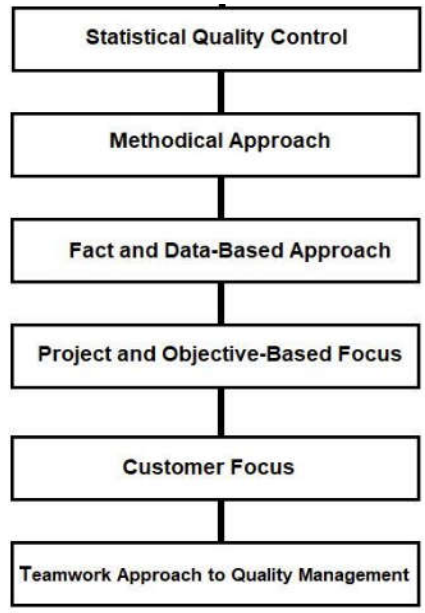


Figure 5. Characteristics of Six Sigma

1. **Statistical Quality Control:** Six Sigma is taken from the Greek letter ?, which in statistics stands for standard deviation. The output quality is evaluated using the standard deviation.

2. **Methodical Approach:** A systematic application method for DMAIC and DMADV called Six Sigma can be used to raise manufacturing quality. Design-Measure-Analyze-Improve-Control, or DMAIC, is an acronym. Design-Measure-Analyze-Design-Verify is also known as DMADV.

3. **Fact and Data-Based Approach:** The statistical and methodical method shows the scientific basis of the technique.

4. **Project and Objective-Based Focus:** The Six Sigma technique is used to concentrate on the circumstances and requirements.

5. **Customer Focus:** The core of the Six Sigma methodology is the customer focus. The standards for quality improvement and control are based on particular customer demands.
6. **Teamwork Approach to Quality Management:** In order to improve quality, firms must organize using the Six Sigma approach.

2.8.2 Six Sigma Methodologies

The methodologies used in Six Sigma projects are as follows:

1. DMAIC:

DMAIC is used to enhance an existing business process and contains five phases:

Define

Measure

Analyze

Improve

Control

2. DMADV:

DMADV is used to create new product designs or process designs and also contains five phases:

Define

Measure

Analyze

Design

Verify

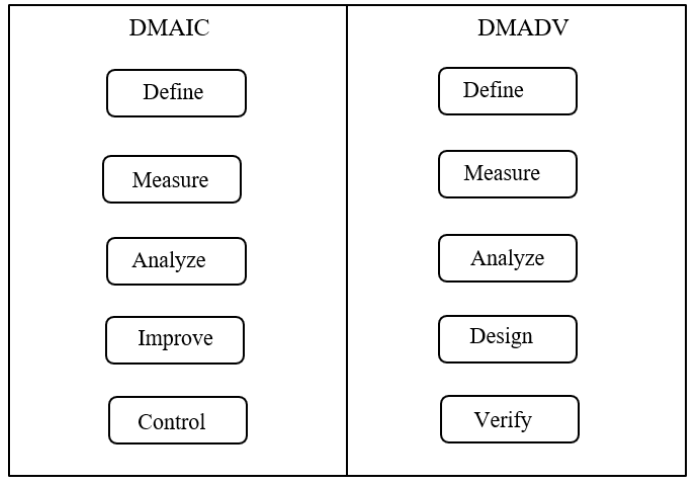


Figure 6. Six Sigma Methodologies

2.9 QUALITY IMPROVEMENT

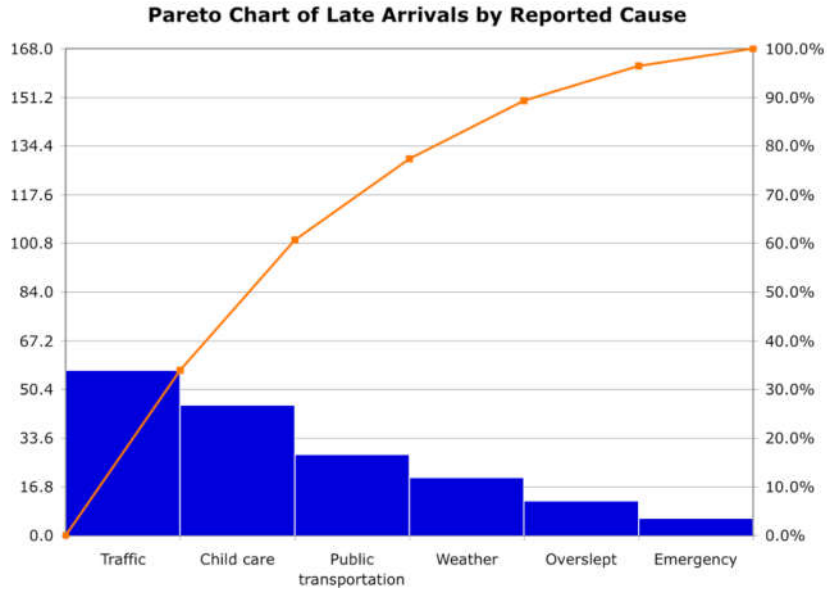
The strategy for methodically raising the standard of care is quality improvement. To eliminate variance, generate predictable results, and enhance outcomes for patients, healthcare systems, and companies, quality improvement aims to standardize processes and structure.

2.9.1. Pareto Chart

The Pareto chart aids in prioritizing the most important issues for remedial action or in focusing on specific problem areas.Based on the Pareto 80-20 rule, the Pareto chart. It suggests that 20% of the few significant causes or factors—often referred to as the Vital Few—are responsible for 80% of the issues or failures.And the remaining 20% of issues are the result of 80% of numerous trivial causes, also known as trivial many. As a result, it provides us with data about the Vital Few from the Trivial Many.One of the crucial basic 7 QC Tools, this tool is widely utilized in problem-solving methodologies like 8D, PDCA, and Six Sigma.

Steps to make a Pareto Chart:

1. Record the data – Refer Check Sheet.
2. Order the data.
3. Label the vertical axis.
4. Label the Horizontal axis.
5. Plot the Bars.
6. Add up the counts.
7. Add a cumulative line.
8. Add title and Legends.
9. Analyze the Chart.
10. Interpret the results.



Benefits of Pareto Chart:

1. It is simple to create.
2. Helps understand a problem.
3. Helps to analyze Weighted cost of problem.

2.9.2 Scatter Diagram

In addition to Scatter Diagram, other names for it include Scatter Plot, Scatter Chart, and Scatter Graph. To determine the link between the two variables, a scatter graph is employed. Data with independent variables are often represented along the horizontal X-axis, while data with dependent variables are typically plotted along the vertical Y-axis. Controlled parameters are another name for an independent variable. A positive or negative correlation between the two variables is displayed. A positive correlation exists if the distribution of the plotted dots is from lower-left to upper-right. It is a negative correlation if the plotted dots are dispersed from the upper left to the bottom right.

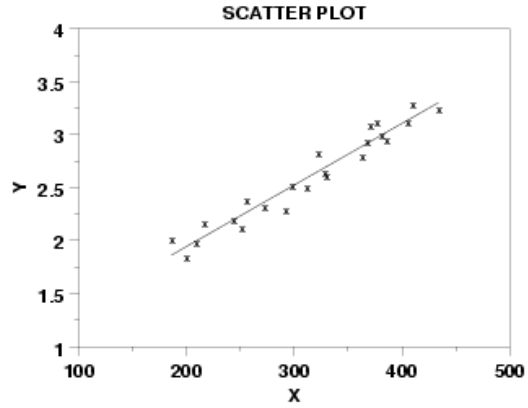
Because a scatter graph has two parameters that represent Cause and Effect, it is comparable to a fishbone diagram. These two, though, are utterly dissimilar.The cause-and-effect link is examined using the fishbone diagram; however this relationship is not shown. In contrast, a scatter plot makes it easier to see how the two variables are related.

Scatter plot correlation types:

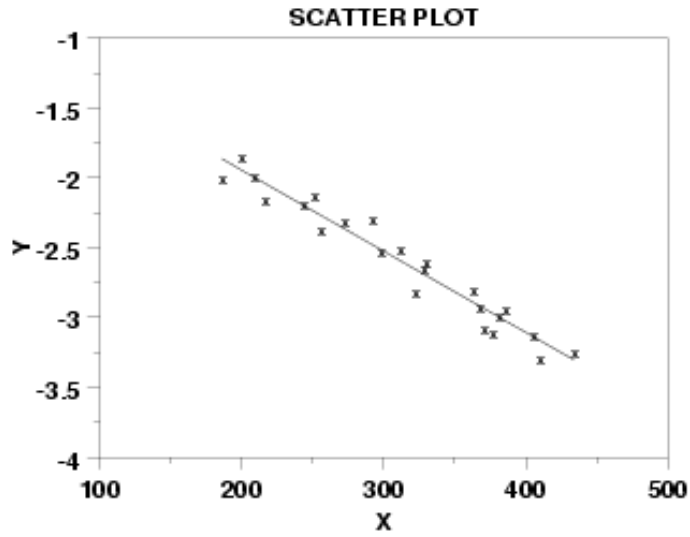
There are five categories of scatter chart representation:

1. Scatter chart with Strong or High Positive Correlation.
2. Scatter chart with Strong or High Negative Correlation.
3. Scatter chart with Week or Low Positive Correlation.
4. Scatter chart with Week or Low Negative Correlation.
5. Scatter chart with Weakest or No Correlation.

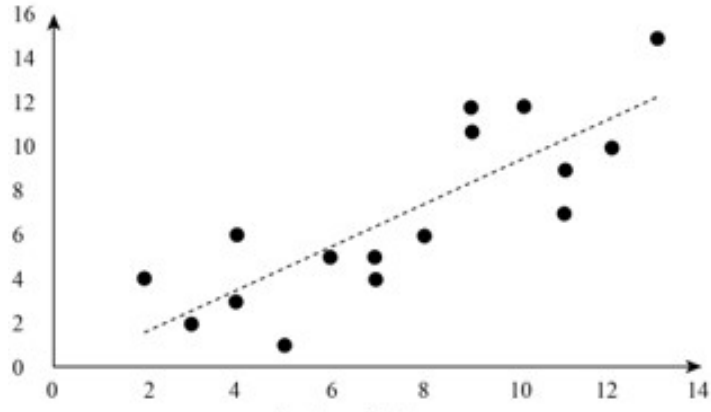
1. **Scatter chart with Strong or High Positive Correlation:** The X values increases and so the Y value also increases.



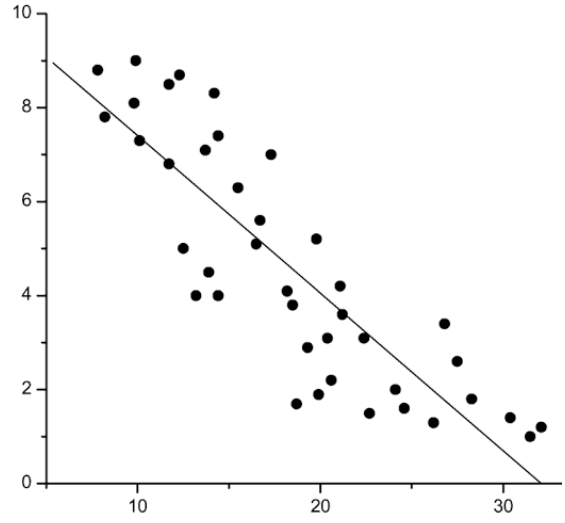
2. **Scatter chart with Strong or High Negative Correlation:** The X values decreases and so the Y value also decreases.



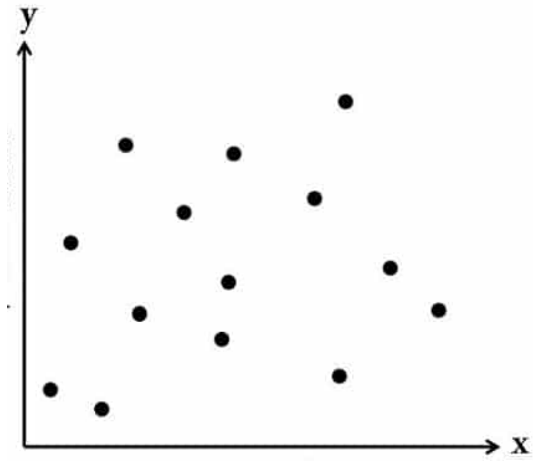
3. **Scatter chart with Week or Low Positive Correlation:** Value of x increases, the value of y slightly increases but not in straight line.



4. **Scatter chart with Week or Low Negative Correlation:** Value of x increases, the value of y slightly decreases but not in straight line.



5. Scatter chart with Weakest or No Correlation has no pattern or unclear relations.

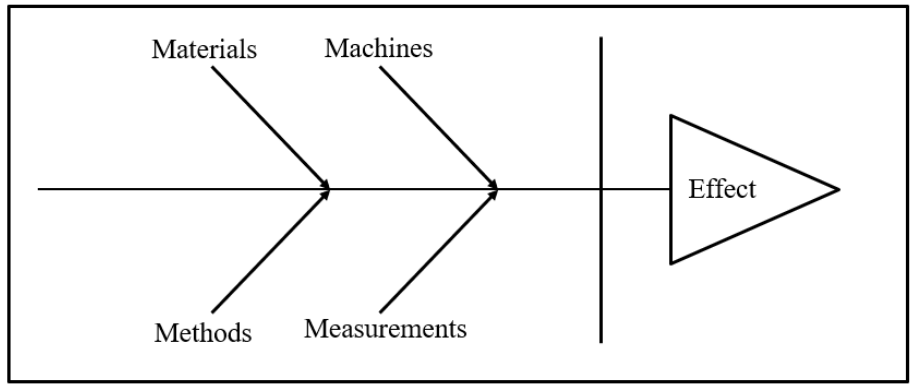


2.9.3 Cause-and-effect diagram:

It was designed by Kaoru Ishikawa. It resembles a skeleton of fish hence it is also known as fishbone diagram. They are used to pinpoint the numerous causes (factors) that contribute to a problem (effect). In the end, it aids in identifying the problem's underlying cause, enabling you to successfully identify the right remedy.

How to use it?

1. Identify the problem area that needs to be studied and note it at th head.
2. Determine the primary causes of the issue. The labels for the fishbone diagram's principal branches are as follows. Methods, materials, equipment, personnel, policies, and procedures are a few examples of these broad categories.
3. Determine logical alternatives to the primary reasons and add them as branches to the main branches.
4. Investigate the major and minor causes more thoroughly using the diagram you produced as a guide.
5. Make an action plan outlining your approach to solving the issue once you have determined the core cause.



Benefits of Fishbone diagram:

1. The emphasis is on causes rather than symptoms or presumptions.
2. Break problems down into little parts to identify the true core cause.
3. Involve more individuals and encourage teamwork.
4. Enhances the efficiency and performance of the team.
5. boosts understanding of the process.

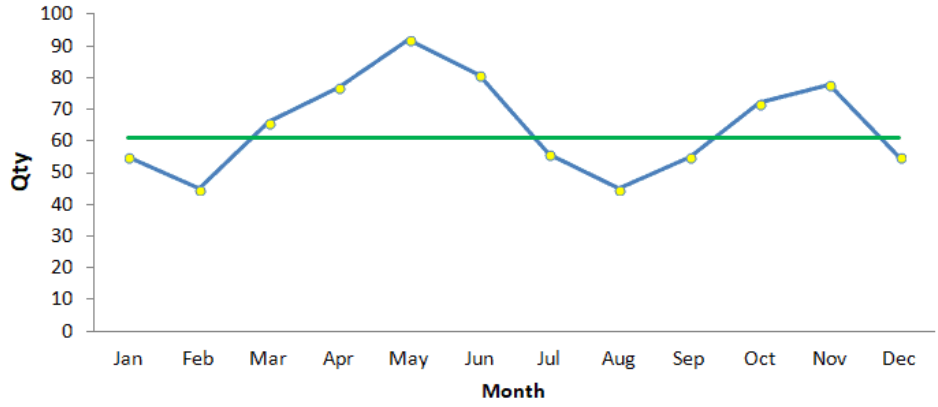
2.9.4 Run Chart

A line graph of data plotted over time is called a run chart. You can uncover trends or patterns by gathering data over time and charting it. Run charts are unable to determine whether a process is stable since they do not employ control limits. They can, however, demonstrate the procedure in action. The run chart can be a useful tool early on in a project since it shows crucial information about a process before you've gathered enough information to establish trustworthy control limits.

Run charts show individual data points in chronological order. Run chart use median value and apply rules for detecting special charts variation.

Creating a run chart:

1. Pick the data to track.
2. Gather data, 20-25 datapoints to check meaningful patterns over time.
3. Make a graph on which you can display your data as a function of time (x axis, or horizontal line), using the y axis as the vertical axis.
4. Plot the data.
5. Interpret the chart.



2.10 QUALITY COSTS

The costs incurred to ensure that your product is of a high caliber are referred to as quality costs. It entails guarding against, spotting, and resolving any product problems. It's essential to make sure that your product lives up to the customer's expectations, which goes beyond simply improving its appearance. For instance, if someone buys a car for very little money, they won't anticipate luxury seats or air conditioning. But they'll be looking for the automobile to perform well. Here, quality is defined as a functional vehicle as opposed to one that has opulent amenities.

Companies must carefully measure and manage their quality costs since they can have a big influence on their bottom line.Analysis of software quality costs can focus SQA efforts on improving activities that frequently fail and have high failure costs, depending on the specific SQA technique being used, or, alternatively, on improving expensive control activities. This analysis helps others learn from them and replicate their success by focusing on the teams whose efforts keep their quality expenses much below the average. At the same time, quality cost analysis can assist in identifying teams whose ineffective quality assurance efforts result in higher-than-average quality costs. The outcomes can then be used to aid in team development.

2.10.1 Types of quality costs.

There are 6 types of quality costs.

1. **Costs of prevention:** Costs associated with preventing errors, such as those associated with instructing and training the maintenance personnel as well as with taking preventative and corrective measures.

2. **Costs of appraisal:** Costs associated with mistake detection, including those associated with external teams, SQA teams, and customer satisfaction surveys' reviews of maintenance services.

3. **Costs of managerial preparation and control:** Costs associated with administrative actions taken to prevent errors, such as those associated with creating maintenance plans, hiring a maintenance crew, and monitoring maintenance performance.

4. **Costs of internal failure:** Costs associated with software failure fixes that the maintenance staff started (before hearing from customers).

5. **Costs of external failure:** Costs associated with fixing software errors brought on by consumer complaints.

6. **Costs of managerial failure:** Expenses of software failures brought on by managerial decisions or inactivity, i.e., expenses of damages brought on by a lack of maintenance staff and/or an improper division of maintenance tasks.

2.10.2 Quality Cost Measurement

Each business calculates its cost of quality differently. Organizations frequently calculate the entire warranty costs as a proportion of sales to measure the cost of quality. The Cost of Quality is examined externally rather than internally by this method. A more thorough breakdown of all quality costs is necessary for comprehension.

Cost of Quality (COQ) is calculated by adding COGQ and COPQ,

COGQ = Cost of Good quality

COPQ= Cost of Poor quality

$$COQ = COGQ + COPQ$$

Cost of Quality is categorized by Prevention, Appraisal, Internal Failure, and External Failure.

The Cost of Good Quality is the total of Prevention Cost and Appraisal Cost (COGQ = PC + AC).

The Cost of Poor Quality is the addition of Internal and External Failure Costs (COPQ = IFC + EFC)

Hence, COQ can also be given by:

$$COQ = (PC+AC) + (IFC+EFC)$$

2.10.3 Quality Cost in decision making.

The organization should achieve financial success through all its endeavours. Costs associated with quality can be used to support improvements made to a good or service. Typically, the sponsor must decide which initiatives will provide the maximum return on investment before making investments in new machinery, supplies, or facilities. The capacity to produce a wider variety of higher-quality items as well as labour and manufacturing time savings are virtually usually considered in these calculations. When analysing quality costs, especially those connected to faults, the "higher quality" factor can be put into numbers. It is crucial to calculate the costs associated with inspecting materials from the time they are received through processing, organization, repair, and trash, as well as the intangible expenses related to the delivery of non-compliant goods or services to the client.

It's crucial to calculate the expenses of inspecting materials from the time of their reception through their processing, organization, repair, and disposal, as well as the intangible costs related to the delivery of non-compliant goods or services to the client. Determining the true profitability of a product or service requires making judgments based on more thorough quality information, such as costs associated with product evaluation.

Project reviewers can decide whether project funds represent a worthwhile expenditure that will aid in the company's expansion once the quality expenses have been established. It is advantageous to first identify quality expenses and then quantify them since both quality and possible cost savings are revealed. The quality costs of a corporation are improved by maximizing its quality performance. Regardless of the kind of quality measurement or quality improvement system used, all subsequent activity must be informed by the knowledge and improved.

The Plan-Do-Study-Act circle proposed by Edward Deming is crucial in this context. The organization should take action once the enhancements have been designed and put into place to guarantee that future operations will continue to operate at the new level of performance and at the same level of lower quality costs. Managers can evaluate improvement investments and their contributions to profits by determining quality costs. The value of quality programs is directly correlated with how well they can increase customer satisfaction and, eventually, the bottom line for the business.

2.11 LET US SUM UP

In this unit we saw the importance of software quality. SQA shouldn't be kept to just the development stage. Instead, it should be expanded to include the lengthy years of service that follow the delivery of the product. Incorporating quality concerns that are specifically relevant to the software product expands the concept of SQA by incorporating software maintenance activities.

We saw different charts and diagrams to improve the quality. The software quality cost is also an important concept and it is not different from other costs. Quality costs appear in each of the stages of the product life cycle, as well as at all the operational levels of the company

2.12 LIST OF REFERENCES

1. Galin, D., 2004. Software quality assurance: from theory to implementation. Pearson education.

2. Horch, J.W., 2003. Practical guide to software quality management. Artech House.

3. Laporte, C.Y. and April, A., 2018. Software quality assurance. John Wiley & Sons.

2.13 BIBLIOGRAPHY

1. Duncan, S., 2005. Software Quality Assurance: From Theory to Implementation. Software Quality Professional, 7(3), p.42.

2. Naik, K. and Tripathy, P., 2011. Software testing and quality assurance: theory and practice. John Wiley & Sons.

2.14 UNIT END EXERCISES

1. Explain Software Quality Assurance (SQA) in brief?
2. What are the characteristics of the Six Sigma rule?
3. Calculate the cost of quality (COQ) if appraisal cost is $10,000, repair cost $15,000.
4. Explain different categories of scatter diagram.
5. What are different types of quality costs? Explain.
6. Explain in brief about ISO 9000.

3

SOFTWARE TESTING STRATEGIES

Unit Structure

3.0 OBJECTIVES

- To get familiar with different strategies involved in software testing
- To get acquaint with the different types of testing involved in software testing approach

3.1 INTRODUCTION: STRATEGIC APPROACH TO SOFTWARE TESTING

Software testing involves assessing a software application to see if it complies with requirements and to spot any flaws. These are typical testing techniques:

1. Black box testing - Tests the software's functionality without examining the internal code layout.
2. White box testing - Examines the software's internal logic and code structure.
3. Unit testing verifies that individual software modules or components are operating as intended.
4. Integration testing verifies that various software components are integrated and function as a system.
5. Functional testing - Verifies that the functional specifications of the software are satisfied.
6. System testing verifies that the entire software system satisfies the required specifications.
7. Acceptance testing verifies that the software satisfies the requirements of the client or end user.
8. Regression testing verifies that the programme has not developed new flaws after updates or adjustments have been done.
9. Performance testing - This checks the software's speed, scalability, and stability to see how it performs.
10. Software is put through security testing to check for flaws and make sure it complies with security standards.

Software testing is a form of investigation to determine if there are any flaws or defects in the software so that they can be fixed to improve the software's quality and determine whether or not it satisfies the criteria.

Glen Myers claims that the following goals of software testing:

- Testing is the process of looking at and examining a programme to determine whether there is an error or not and whether it satisfies the requirements or not.
- A good test case and successful testing are both indicated by a large number of errors that were discovered throughout the test.
- A successful test case will reveal an undiscovered error that hasn't been found yet.

The fundamental goal of software testing is to create the tests in such a way that they quickly and efficiently identify all types of mistakes,

reducing the amount of time needed for software development. The overall plan for software testing entails:

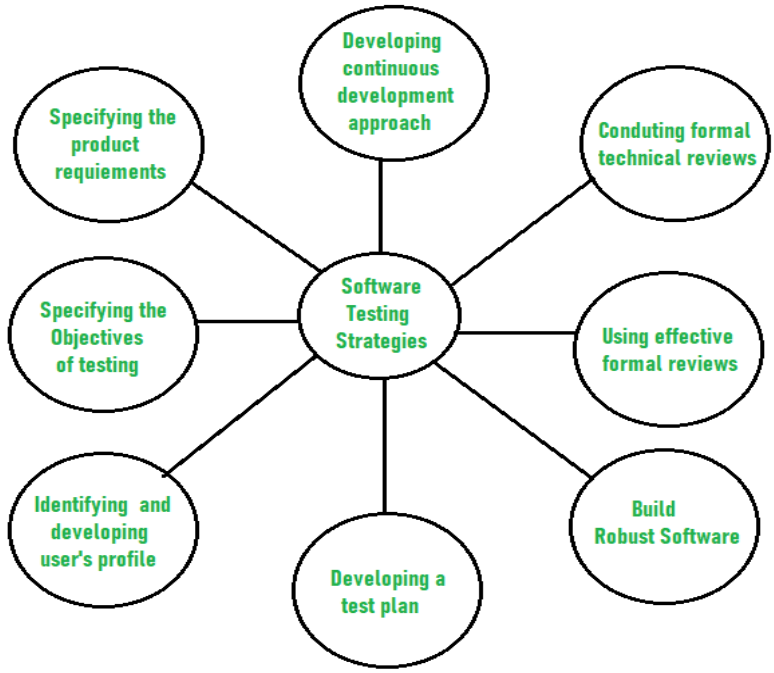


Figure 1: Overall strategy for software testing

1. **Prior to testing, it's important to recognise and precisely define the product requirements in terms of numbers:** There are various qualities of software, such as maintainability, which refers to the capacity to update and alter, likelihood, which refers to the capacity to identify and quantify any risk, and usability, which refers to the ease with which customers or end-users may use it. To ensure accurate test results, all these distinguishing characteristics should be listed in a specific order.

2. **Clearly and specifically stating the testing objectives:** The efficacy of the software's capacity to reach the target, any failure to meet criteria and carry out functions, and the cost of defects or errors, which refers to the expense involved in correcting the error, are a few examples of testing objectives. The test strategy needs to explicitly state each of these objectives.

3. **Classifying users and creating individual user profiles for the software:** Use cases outline how various user classes interact with the system and one another to accomplish a goal. to determine the users' genuine needs, followed by a test of the product's actual use.

4. **Setting value in test planning and concentrating on rapid-cycle testing:** Rapid Cycle Testing is a sort of test that enhances quality by discovering and evaluating any modifications needed to enhance the software development process. Consequently, a test plan is a crucial and useful document that aids the tester in carrying out quick cycle testing.

5. **Is it possible to create reliable software that can test itself:** Different sorts of errors should be able to be found or identified by the software. Additionally, software design should permit automated and regression testing, which examines the software to determine whether any negative or unintended consequences of changes to the code or programme have an impact on its functionality.

6. **Employing efficient formal reviews as a filter before testing:** Formal technical reviews are a method to find errors that haven't yet been found. Effective technical reviews conducted before to testing significantly cut down on the testing workload and testing time, hence speeding up the total software development process.

7. To assess the nature, suitability, or capability of the test strategy and test cases, conduct formal technical reviews. The thorough technical review aids in identifying any gaps in the testing strategy that need to be filled. Therefore, in order to raise the calibre of software, technical reviewers must assess the effectiveness and quality of the test plan and test cases.

8. **Creating a continuous development strategy for the testing process:** To assess and control the quality of software development, a test method that has already been measured as part of a statistical process control approach should be utilised.

- **Advantages of software testing:**

1. **Enhances software quality and dependability -** Testing aids in the early detection and correction of flaws, lowering the possibility of failure or unexpected behaviour in the finished product.

2. **Improves user experience -** Testing aids in detecting usability problems and enhancing the user experience as a whole.

3. **Builds trust -** By testing the programme, stakeholders and developers can feel more certain that it satisfies specifications and performs as intended.

4. **Makes maintenance easier -** Testing makes it simpler to maintain and update the product by locating and fixing bugs early.

5. **Lowers costs** - Finding and resolving flaws early in the development phase saves money over the course of the product's lifespan.

- **Disadvantages of software testing:**

1. Testing helps in the early detection and rectification of defects, minimising the likelihood of failure or unexpected behaviour in the finished product.

2. Enhances user experience - Testing helps identify usability issues and improves the overall user experience.

3. Promotes trust - By putting the programme through testing, stakeholders and developers can be more confident that it meets requirements and works as intended.

4. Makes maintenance simpler - By identifying and resolving defects early on, testing makes it easier to maintain and update the product.

5. Reduces costs - Fixing problems quickly during the development stage reduces costs over the duration of the product's life.

3.2 UNIT TESTING

Each unit or individual component of the software application is tested as part of the unit testing process. It represents the initial stage of functional testing. The purpose of unit testing is to confirm the functionality of individual unit components.

A unit is a single testable component that may be tested as part of the application software development process.

Unit testing is used to ensure that isolated code is correct. A specific application function or piece of code is referred to as a unit component. Unit testing is typically conducted using the white box testing methodology by developers.

When the application is finished and submitted to the test engineer, the test engineer will begin independently or one-by-one testing each component of the module or module of the application. This procedure is referred to as unit testing or components testing.

3.2.1 Why Unit testing?

Unit testing is the first level of testing carried out before integration and further levels of testing in a testing level hierarchy. It employs modules for testing, reducing the reliance on waiting for Unit testing is aided by the usage of stubs, drivers, dummy objects, and unit testing frameworks.

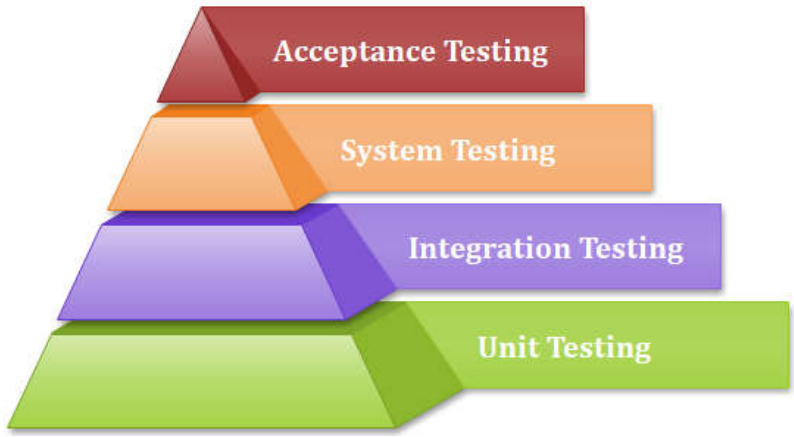


Figure 2: Testing Hierarchies

The software is typically tested at four levels: unit testing, integration testing, system testing, and acceptance testing. However, due to time constraints, software testers occasionally skip unit testing, which can result in higher defects during integration testing, system testing, acceptance testing, or even beta testing, which happens after a software application is finished.

Here are a few essential justifications:

- Unit testing enables developers and testers to swiftly update code that is producing defects by assisting them in understanding the fundamentals of the program.
- It helps to have unit tests for documentation.
- There is a chance that there will be fewer flaws in subsequent testing levels because unit testing catches errors relatively early in the development process.
- By relocating code and test cases, it promotes code reuse.

3.2.2 Unit testing tools

We have various types of unit testing tools available in the market, which are as follows:

- NUnit
- JUnit
- PHPunit
- ParasoftJtest
- EMMA

4.2.3 Unit testing techniques

Unit testing uses all white box testing techniques as it uses the code of software application:

- Data flow Testing
- Control Flow Testing
- Branch Coverage Testing
- Statement Coverage Testing
- Decision Coverage Testing

3.2.4 How to achieve best results using Unit testing

By taking the actions outlined below, unit testing can produce the best results without creating confusion or adding complexity:

- As the test cases won't be impacted by requirement changes or enhancements, test cases must be independent.
- Unit test cases must have clear and consistent naming conventions.

- Before moving on to the next stage of the SDLC, the defects found during unit testing must be rectified.
- One code should only ever be tested at once.
- Adopt test cases as you write the code; otherwise, the number of possible execution pathways would rise.
- Verify whether the matching unit test is accessible or not for any module whose code has changed.

3.2.5 Advantages of Unit testing

- Unit testing employs a modular approach because any component can be tested without holding up the testing of other components.
- To comprehend the unit API, the developing team focuses on the functionality that is offered by the unit and how functionality should appear in unit test cases.
- After a few days, unit testing enables the developer to modify code and verify that the module is still operating faultlessly.

3.2.6 Disadvantages of Unit testing

- As it only examines individual code units, it cannot detect integration or high-level errors.
- Since it is impossible to evaluate every execution route during unit testing, errors in programs cannot be found in every instance.
- It works best when combined with other diagnostic procedures.

3.3 INTEGRATION TESTING

After unit testing, the software testing process moves on to integration testing. Units or individual software components are tested collectively during this testing. The goal of the integration testing level is to identify flaws when integrated components or units interact.

Modules are used in unit testing for testing purposes, and integration testing combines and tests these modules. The Software is created using a variety of software modules that were created by various programmers or coders. Integrity testing is done to ensure that all of the modules are communicating properly.

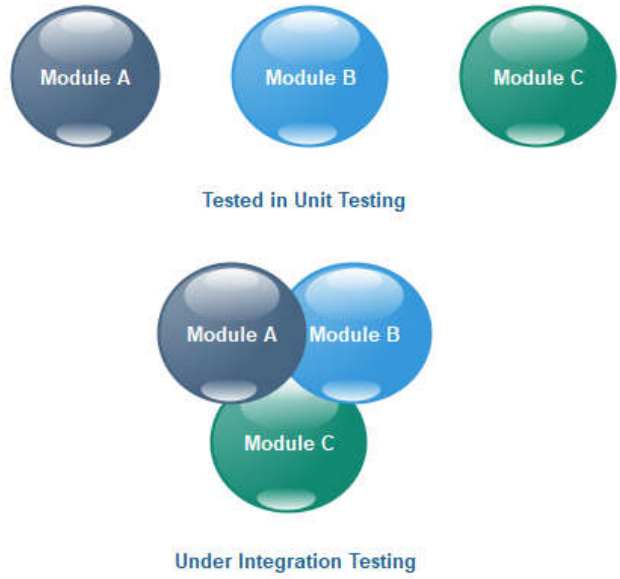


Figure 3: Integration Testing

Integration testing is the process of verifying the data flow between dependent modules when each component or module is functioning independently.

3.3.1 Guidelines for Integration Testing

- After each application module has undergone functional testing, we only go on to integration testing.
- In order to ensure that a suitable sequence is followed and that we don't miss any integration cases, we always perform integration testing by selecting modules one at a time.
- Determine the test case strategy first, which will help you create executable test cases based on the test data.
- Examine the application's structure and architecture to determine the most important modules to test first and to discover all potential situations.
- Create test cases to thoroughly verify each interface.
- Select input data before running the test case. Testing heavily relies on the input data.
- If we discover any bugs, we should notify the developers, who will subsequently repair the issues and retest.
- Test integration both positively and negatively.

If the entire balance is Rs. 15,000 and we are sending Rs. 1500, then this positive testing means that the amount transfer should be successful. The test would be considered successful if it did.

Negative testing, on the other hand, means that if the total balance is Rs. 15,000 and Rs. 20,000 is being transferred, the test will pass if neither event occurs. If it does, there is a code bug, and we will communicate it to the development team so they can repair it.

3.3.2 Reason behind Integration Testing

Even though unit testing was performed on every module of the software application, errors still exist for the following reasons:

1. Integration testing is crucial to determining how well software modules function because each module is created by a different software developer, whose programming logic may differ from that of developers of other modules.
2. to determine whether or not the software modules' interactions with the database are accurate.
3. At the time a module is being developed, requirements can be modified or improved. Integration testing is now required because it's possible that these additional requirements won't be tested at the unit testing level.
4. Errors may be caused by software module incompatibility.
5. to check whether hardware and software are compatible.
6. Inadequate exception handling between modules can lead to issues.

3.3.3 Integration Testing Techniques

Any testing technique (Blackbox, Whitebox, and Greybox) can be used for Integration Testing; some are listed below:

Black Box Testing

- State Transition technique
- Decision Table Technique
- Boundary Value Analysis
- All-pairs Testing
- Cause and Effect Graph
- Equivalence Partitioning
- Error Guessing

White Box Testing

- Data flow testing
- Control Flow Testing
- Branch Coverage Testing
- Decision Coverage Testing

3.3.4 Types of Integration Testing

Integration testing are categorized into two types: Incremental and Non-Incremental

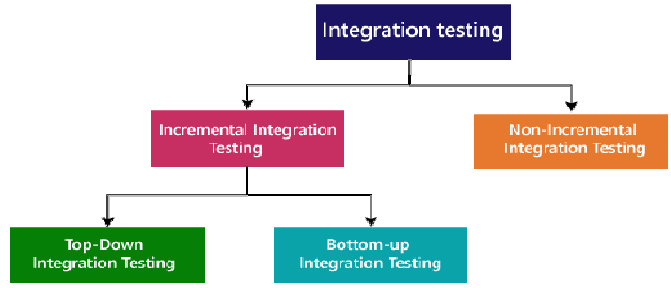


Figure 4: Types of Integration testing

In the incremental approach, modules are added one at a time in ascending sequence or as needed. The modules you choose must make sense together. Usually, two or more modules are added and tested to see if the functionalities are correct. Up until all of the modules have undergone successful testing, the procedure continues.

OR

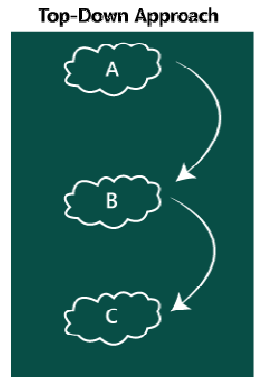
The dependent modules in this kind of testing have a close link with one another. Let's say we test the proper operation of the data flow between two or more modules. If so, try again after adding more modules.

Incremental integration testing is carried out by further methods:

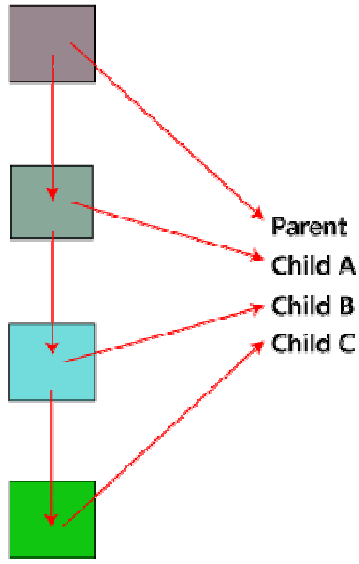
- Top-Down approach
- Bottom-Up approach

Top-Down approach

The top-down testing technique focuses on the process in which lower-level modules are tested alongside higher-level modules until all of the modules have been successfully tested. Critical modules are tested first, allowing for the early detection and correction of significant design problems. With this approach, the modules will be added gradually or one at a time, and the data flow will be examined in the same sequence.



In the top-down approach, we will be ensuring that the module we are adding is the child of the previous one like Child C is a child of Child B and so on as we can see in the below image:



Advantages:

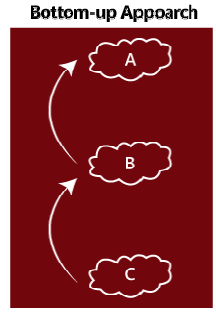
- Identification of defect is difficult.
- An early prototype is possible.

Disadvantages:

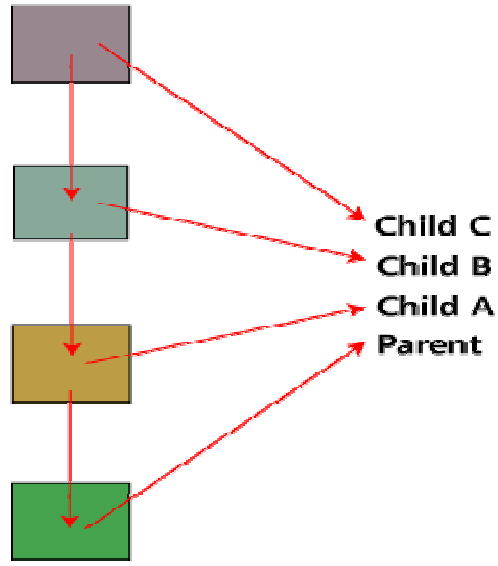
- Due to the high number of stubs, it gets quite complicated.
- Lower level modules are tested inadequately.
- Critical Modules are tested first so that fewer chances of defects.

Bottom-Up approach

The technique of testing lower level modules with higher level modules until all of the modules have been successfully tested is known as the "bottom to up" testing strategy. Since top-level critical modules are tested last, a defect may result. Another option is to indicate that we will install the modules in order, starting at the bottom, and then examine the data flow.



In the bottom-up method, we will ensure that the modules we are adding are theparent of the previous one as we can see in the below image:



Advantages

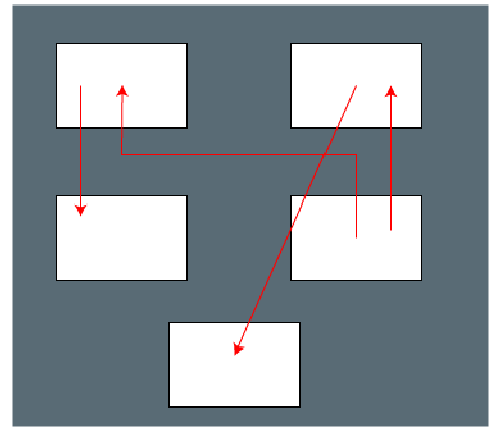
- Identification of defect is easy.
- Do not need to wait for the development of all the modules as it saves time.

Disadvantages

- Critical modules are tested last due to which the defects can occur.
- There is no possibility of an early prototype.

Non-Incremental Integration testing:

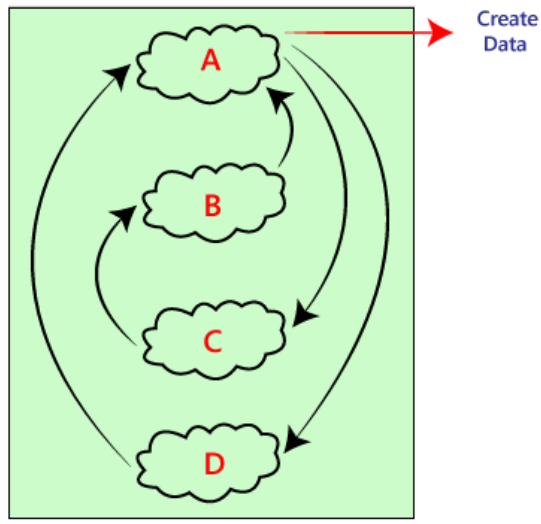
When the data flow is exceedingly complicated and it is challenging to determine who is a parent and who is a child, we will use this method. And in this situation, we will generate the data in any module and then check to see if it is present in all other existing modules. Consequently, the Big Bang approach is another name for it.



Big bang Method:

This method integrates all components at once and then conducts testing. While it is practical for small software systems, it makes it challenging to identify flaws in large software systems.

Since the testing team has less time to execute this process because this testing might be done after all modules have been completed, internal connected interfaces and high-risk important modules are more likely to be overlooked.



Advantages:

- It is practical for compact software systems.

Disadvantages:

- Defect identification is challenging since we are unable to determine the source of the error, making it impossible to determine where the fault originated.
- Small modules are readily missed.
- There is extremely little time allotted for testing.
- There's a chance we won't test all of the interfaces.

3.4 VALIDATION TESTING

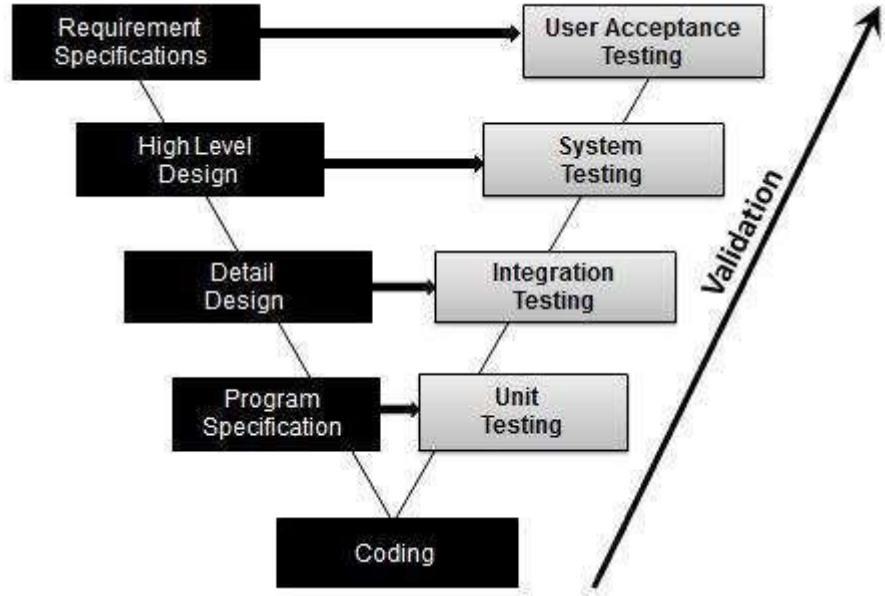
The procedure of assessing software to see if it satisfies stated business requirements either during development or at the end of development.

Validation testing makes ensuring that the product really does meet the needs of the customer. It is also possible to describe it as proving that a product works as intended when used in the right setting.

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

Workflow of Validation Testing:

Validation testing can be best demonstrated using V-Model. The Software/product under test is evaluated during this type of testing.



Activities:

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

3.5 SYSTEM TESTING

Testing a fully integrated software system is part of system testing. Typically, software is integrated into computer systems (software itself is just one component of a computer system). To create a comprehensive computer system, the software is developed in modules and then interfaced with hardware and other applications. To put it another way, a computer system consists of a collection of software that can carry out a variety of functions, but only software can do so since it needs to communicate with appropriate hardware. System testing is a collection of several types of tests designed to put an integrated software computer system through its paces and check it against requirements.

System testing is the process of examining an application's or piece of software's overall usability. We test the product as a whole system and travel (go through) all the required modules of an application to see if the final features or the final business function as intended.

End-to-end testing is done in an environment that is comparable to the one used in production.

Black box testing, which includes testing the software's external functionality, includes system testing. To find little flaws, testing mimics the user's perspective.

The following actions are a part of system testing.

- Verification of the application's input features to see if it is generating the desired results.
- Integrated software is tested with external devices to see how different parts interact with one another.
- System-wide testing for end-to-end testing.
- Using auser's expertise, test the application's behaviour

3.5.1 Types of System testing

There are more than 50 different forms of system testing, but software testing businesses typically use a few of them. Here are some of them:

1] Regression Testing

Regression testing is carried out as part of system testing to check and pinpoint any defects that may have arisen as a result of changes made to any other system component. It ensures that any modifications made throughout the development process did not produce a new flaw and ensures that no existing defects will be present as new software is added over time.

2] Load Testing

To determine whether the system can function under real-time loads, load testing is conducted alongside system testing.

3] Functional Testing

A system is put through functional testing to see whether there are any missing functions. The tester compiles a list of crucial features that should be included in the system, can be introduced during functional testing, and should raise the system's quality.

4] Recovery testing:

System testing includes recovery testing of a system, which verifies the system's dependability, credibility, accountability, and ability to recover. It ought to be able to successfully recover from all potential system failures.In this testing, we will evaluate the application's ability to bounce back from errors or natural calamities.

5] Migration Testing

Migration testing is performed to ensure that if the system needs to be modified in new infrastructure so it should be modified without any issue.

6] Usability Testing

The purpose of this testing to make sure that the system is well familiar with the user and it meets its objective for what it supposed to do.

7] Software and Hardware Testing

The system will be tested to see if the hardware and software are compatible. To operate the software without any problems, the hardware setup must be compatible. Because it allows for interactions between hardware and software, compatibility promotes flexibility.

3.5.2 Why is System testing important

- System testing, which examines the entire system's functionality, provides total assurance of system performance.
- It comprises testing of the business requirements as well as the system software architecture.
- Even after production is complete, it aids in bug and live issue mitigation.
- System testing feeds the same data into both an old and a new system, comparing the functional changes between the two so that the user may appreciate the advantages of the system's newly added features.

3.6 SUMMARY

Software testing involves assessing a software application to see if it complies with requirements and to spot any flaws. We have covered seven

different steps involved in software testing strategies. We have also focused on different types of testing along with their fundamentals. We have dwelled into integration testing, unit testing and system testing along with the concepts associated with them.Each unit or individual component of the software application is tested as part of the unit testing process. In integration testingunits or individual software components are tested collectively.Validation testing makes ensuring that the product really does meet the needs of the customer

3.7 LIST OF REFERENCES

1. Software Engineering for Students, A Programming Approach, Douglas Bell, 4th Edition, Pearson Education, 2005
2. Software Engineering – A Practitioners Approach, Roger S. Pressman, 5th Edition, Tata McGraw Hill, 2001
3. Quality Management, Donna C. S. Summers, 5th Edition, Prentice-Hall, 2010. 3. Total Quality Management, Dale H. Besterfield, 3rd Edition, Prentice Hall, 2003.

3.8 UNIT END EXERCISES

1. Explain the concept of Unit Testing.
2. State the unit testing tools and techniques.
3. Illustrate the Advantages, Disadvantages and the way to achieve best results using Unit testing.
4. Write a detailed note on Integration Testing.
5. State different Reason behind Integration Testing
6. Explain Integration Testing Techniques.
7. Explain the Types of Integration Testing.
8. Write a note on Validation testing.
9. Explain the concept of SystemTesting.
10. Write a note on thetypes of System testing.
11. Explainwhy is System testing important.

4

SOFTWARE METRICS

Unit Structure :

4.0 OBJECTIVES

- To get familiar with the concepts and development metrics
- To understand the concept of defects and factors associated with defect management

4.1 INTRODUCTION: CONCEPT AND DEVELOPING METRICS

Software testing metrics are quantitative measures used to assess the effectiveness, efficiency, and advancement of the software testing process. This helps us increase the effectiveness of the software testing process and

gather trustworthy data about it. Developers will then be able to plan ahead and make accurate decisions for upcoming testing procedures.

Metric in software testing: A system's or its constituent parts' retention of a specific attribute is measured by a metric. A metric is not defined by testers merely for the purpose of documentation. In software testing, it has many benefits. Developers can use a measure, for instance, to estimate how long it takes to develop software. It may also be used to count the number of new features, improvements, etc., that have been made to the software.

4.2 DIFFERENT TYPES OF METRICS

Software testing metrics come in three different forms:

1. **Process Metrics:** Project features and execution are described by process metrics. These qualities are crucial for the ongoing maintenance and process improvement of the SDLC (Software Development Life Cycle).

2. **Product Metrics:** A product's size, design, performance, quality, and complexity are all determined by its product metrics. Developers can improve the caliber of their software development by utilizing these traits.

3. **Project Metrics:** A project's overall quality is determined by its project metrics. It is used to estimate a project's resources and deliverables as well as costs, productivity, and problems.

Finding the proper testing metrics for the process is of utmost importance. A few things to think about are:

- Prior to creating the metrics, carefully select your target audiences.
- Describe the rationale for creating the measurements.
- Prepare metrics by taking into account the unique project requirements Assess the financial benefit associated with each metric
- Match the measurements to the project lifestyle phase that yields the best results.

4.3 COMPLEXITY METRICS

Cyclomatic complexity is a software metric used to measure the complexity of a program.

A software metric called cyclomatic complexity offers a numerical assessment of the logical difficulty of a programme. The value computed for cyclomatic complexity defines the number of independent paths in the basis set of a programme when used in the context of the basis path testing method. It also gives you an upper bound on how many tests must be run to guarantee that each statement has been executed at least once.

Graph theory serves as the basis for cyclomatic complexity, which offers you a very helpful software metric. One of three methods is used to compute complexity:

1] The number of regions of the flow graphs corresponds to cyclomatic complexity

2]Cyclomatic complexity V(G) for a flow graph G is defined as

$V(G) = E - N + 2$

Where,

E: Number of flow graph edges

N: Number of flow graph nodes

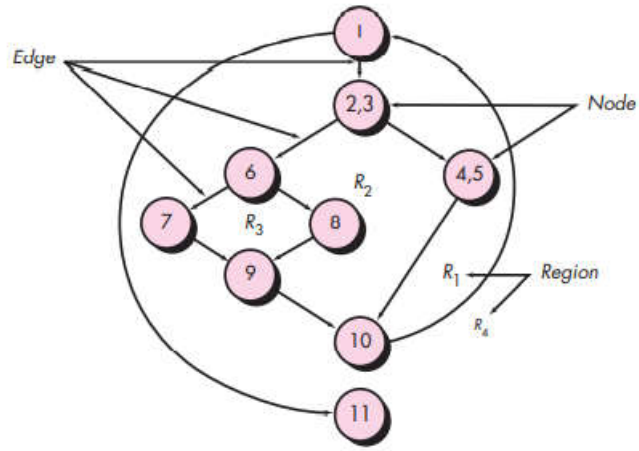
3]Cyclomatic complexity V(G) for a flow graph G is also defined as

$V(G) = P + 1$

Where,

P: number of predicate nodes contained in the flow graph G

For example: Consider the flow graph as shown in the following figure



For this figure the cyclomatic complexity can be computed using each of the algorithms just noted

1] The flow graph consists of 4 regions

2]V(G) = 11 edges - 9 nodes + 2 = 4

3]V(G) = 3 predicate nodes + 1 = 4

Therefore, the flow graph in Figure has a cyclomatic complexity of 4.

More importantly, the value for V(G) gives you an upper constraint on the number of independent routes that make up the basis set, and thus, an

upper bound on the number of tests that must be created and run to ensure that every program statement is covered.

4.4 DEFECT MANAGEMENT: DEFINITION OF DEFECTS

When the predicted outcome differs from the actual result, a software error occurs. Additionally, it could be a bug, weakness, failure, or fault in a computer program. Most defects are the result of faults and blunders made by architects and developers.

The following strategies are used to stop programmers from adding defects during development:

- Adopted programming techniques
- Methodologies for Software Development
- a peer review
- Code Examination

Common Types of Defects

Following are the common types of defects that occur during development:

- Arithmetic Defects
- Logical Defects
- Syntax Defects
- Multithreading Defects
- Interface Defects
- Performance Defects

4.5 DEFECT MANAGEMENT PROCESS

The cornerstone of software testing is the defect management procedure. The most important task for any organization to do when faults have been found is to manage them. This applies to the testing team as well as to everyone else participating in the software development or project management process.

As well-known, reducing the number of defects is best accomplished by defect prevention. Defect prevention is a very economical method for correcting flaws found in earlier phases of software development.The majority of organizations handle Defect Discovery, Defect Removal, and then Process Improvement through the Defect Management Process.The

Defect Management Process (DMP), as the name implies, controls defects by only identifying and correcting the errors.

While it is hard to completely eliminate errors or flaws from software, many problems can be reduced by correcting or resolving them.

The primary goals of the defect management process are to prevent defects, identify defects at the earliest stages, and moderate the impact of defects.

4.5.1 Objective ofDefect Management Process

The following is an overview of the defect management method' primary objective:

- DMP's main goal is to reveal flaws at an early stage of the software development process.
- The execution of the defect management method will assist us in improving the procedure and software implementation.
- The impact or effects of software problems are lessened via the defect management method.
- The DMP, or defect management process, aids in defect prevention.
- Resolving or correcting problems is the primary objective of the defect management process.

The following are the critical objectives of the defect management process for various organizations or projects:

- We are able to contribute to status and progress reports on the defect through the defect management procedure.
- to identify the root cause of the fault and determine the best course of action.
- to offer suggestions and information about the disclosure of the flaw.

4.5.2 Various stages of Defect Management Process

Various stages of Defect Management Process are as depicted in the figure 1 below

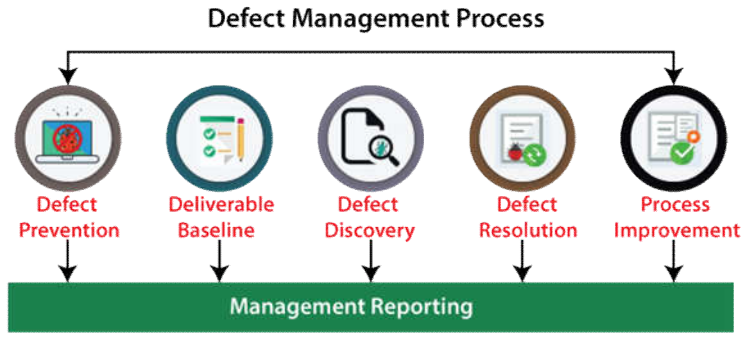


Figure 1:Stages of Defect Management Process

1. **Defect Prevention:** Defect prevention is the first step in the defect management process. The danger of defects is reduced in this stage by following processes, methodology, and accepted practices. The best method for minimizing the impact of a defect is to remove it within the original development stage.Because it is less expensive and the impact can be lessened in the initial stages of addressing or resolving faults.But for later stages, finding flaws and then resolving them can be expensive, and the impacts of a problem might even be exacerbated.

   The defect prevention stage includes

- **Estimate Predictable Impact**: In this step, if the risk is encountered, then we can calculate the estimated financial impact for every critical occasion

- **Minimize expected impact**: When all the critical risk has been discovered, we can take the topmost risks that may be dangerous to the system if encountered and try to diminish or eliminate it.Those risks that cannot be removed will decrease the possibility of existence and its financial impact.

- **Identify Critical Risk**: In defect prevention, we can quickly identify the system's critical risks that will affect more if they happened throughout the testing or in the future stage.

2. **Deliverable baseline:** The Deliverable baseline is the second step in the defect management process. Here, the system, documentation, or product are defined by the delivery.When a deliverable hit its predetermined milestone, we can state that it is a baseline. The deliverable is transported from one step to the next during this phase, and any existing system flaws advance to the subsequent step or milestone. In other words, we may claim that any further changes are controlled once a deliverable is baselined.

3. **Defect discovery:** Defect discovery is the following step in the defect management process. Defect finding is crucial at this early stage of the defect management procedure. Additionally, it could result in longer-term harm.Only a flaw is regarded as detected if developers have acknowledged or recorded it as a valid one.We now know that it is virtually difficult to remove every flaw from a system and make it defect-free. However, we can find the flaws before they cost the project money.

   Identifying a defect, reporting a defect and acknowledging a defect are the phases involved in defect discovery stage.

4. **Defect resolution:** We proceed to the following level of the defect management process, Defect Resolution, when the defect detection stage has been successfully completed.The Defect Resolution method is helpful in identifying and tracking flaws since it outlines a step-by-step process for repairing defects.Giving the development team the

flaws is the first step in this approach. The developers must go forward with the defect's resolution and prioritize fixing it.The developer notifies the testing team of the defect's selection and resolution by sending a defect report.The communication of the test engineer once the issue has been fixed is another step in the defect resolution process.

We need to follow the below steps in order to accomplish the defect resolution stage.

- Prioritize the risk
- Fix the defect
- Report the Resolution

5. **Process improvement:** The previous stage (defect resolution) involved organizing and fixing the defects.We will now examine the lower priority issues because they are still crucial and have an impact on the system during the process improvement phase.From the standpoint of the process improvement phase, all acknowledged faults are equivalent to significant defects and must be corrected.

   The individuals participating in this stage must remember and confirm the source of the defect.Depending on that, we can alter the validation process, base-lining document, and review process to potentially uncover faults early on and reduce the cost of the procedure.These tiny flaws help us figure out how to improve the procedure and get rid of any flaws that could lead to system or product failure in the future.

6. **Management reporting:** The process of defect management ends with management reporting. It is a crucial and important step in the defect management procedure. In order to increase the defect management process and ensure that the generated reports have an aim, management reporting is necessary.

   The evaluation and reporting of defect information, put simply, supports project management, process improvement, and organisation and risk management.The project teams' information gathering on individual problems is the foundation of the management reporting. As a result, each organisation must take into account the data obtained throughout the defect management process and the classification of individual problems.

4.5.3 Defect workflow and states

Many organizations use a technology to perform software testing that records defects throughout the bug/defect lifecycle and also includes defect reports.

At each stage of the defect lifecycle, there is typically one owner who is in charge of completing the necessary tasks to advance the defect report to the next stage.

If we encounter the following circumstance, a defect report may occasionally not have an owner in the latter stages of the defect lifecycle:

- The defect report is cancelled if the defect is invalid.
- If the problem won't be rectified as part of the project, the defect report is regarded as delayed.
- If the fault can no longer be found, the defect report is deemed to be non-reproducible.
- If the issue has been resolved and tested, the defect report is deemed to be closed.

Defect states: If defects are identified throughout the testing, the testing team must manage them in the following three states:

1. **Initial state:** It is the initial defect state, also referred to as the open state.The task of gathering all the information needed to correct the flaws in this stage falls to one or more test engineers.
2. **Returned state:** Return state is the second defect state. In this case, the individual receiving the test report rejects it and requests more details from the report's author.In a returned state, the test engineers have the option of adding more details or accepting the report's rejection.The test manager should check for errors in the initial information collection process itself if several reports are denied.The rejection state or clarification state are other names for the returned state.
3. **Confirmation state:** The test engineer conducted a confirmation test to ensure that the defect had been repaired before reaching the last state of defect, known as the confirmation state.It is accomplished by doing the same actions that revealed the flaw during testing.The report is finished if the flaw is fixed.And if the problem was not fixed, the complaint was reopened and sent back to the owner who had originally saved the defect report for correction.A resolved or verified state is another name for a confirmation state.

4.5.4 Advantages of Defect Management Process

1. **Confirm resolution:** We can ensure that faults are resolved while still being tracked with the use of the defect management process.
2. **Accessibility of automation tools:** The defect or bug tracking process is one of the most important steps in the defect management process.We have a variety of automated technologies for defect monitoring on the market that can assist us in tracking the fault in its early phases.These days, a wide range of tools are available to track various defect kinds. Example:

- **Software tools:** These tools are used to locate or monitor non-technical issues.

- **User-facing Tools:** These kinds of tools will assist us in finding production-related flaws.

3. **Offer valuable metrics:** Along with useful defect metrics, the defect management process also provides us with automation tools.Additionally, these useful defect data support us in reporting and ongoing improvements.

4.5.5 Disadvantages of Defect Management Process

1. If the defect management process is not carried out properly, we risk losing consumers, losing money, and damaging the reputations of our brands.
2. If the defect management process is not handled correctly, there will be a significant amplified cost in the form of a creeping increase in the product's price.
3. Defects may later result in greater harm, and the expense of fixing them will also rise, if they are not handled correctly at an early stage.

4.6 DEFECT REPORTING

A defect report is a document that provides concise information about the faults found, the activities that cause the defects to manifest, and the expected outcomes in place of the application manifesting a defect (error) when doing the specified actions step by step.

Both the Quality Assurance team and the end-users (customers) typically produce defect reports. Since most users test out every element of a program out of curiosity, consumers frequently find more flaws and report them to the software development team. You are now aware of what a defect and a defect report are.

Purpose of creating a report and what to do with them:

Defect reports are made in order to facilitate developers' ability to quickly identify and correct faults. A developer is often given a defect report by QA, reads it, and uses the action steps in the report to replicate the flaws on the software product. The developer then corrects the errors to get the desired result outlined in the report.

Defect reports are crucial and carefully crafted because of this. Defect reports should be concise, well-organized, and to the point. They should also include all the information a developer needs to reproduce the reported faults, as well as the steps taken to identify them. It is usual for QA teams to get defect reports from the clients that are either too short to reproduce and rectify or too long to understand what actually went wrong.

For example,

Defect Description: The application doesn't work as expected.

Now, how in the world does a developer or QA know what went wrong which doesn't meet the client expectation?

In such a case, the developer report to the QA that he couldn't find any problem or he may have fixed any other error but not the actual one client detected. So that's why it's really important to create a concise defect report to get bugs fixed.

A typical defect report contains the information in an xls Sheet as follows.

1. **Defect ID:** A serial number of defects in the report.
2. **Defect Description:**A short and clear description of the defect detected.
3. **Action Steps:**What the client or QA did in an application that results in the defect. Step by step actions they took.
4. **Expected Result:**What results are expected as per the requirements when performing the action steps mentioned.
5. **Actual Result:**What results are actually showing up when performing the action steps.
6. **Severity:**Trivial (A small bug that doesn't affect the software product usage).

1. **Low:** A small bug that needs to be fixed and again it's not going to affect the performance of the software.
2. **Medium:** This bug does affect the performance. Such as being an obstacle to do a certain action. Yet there is another way to do the same thing.
3. **High:** It highly impacts the software though there is a way around to successfully do what the bug cease to do.
4. **Critical:** These bugs heavily impacts the performance of the application. Like crashing the system, freezes the system or requires the system to restart for working properly.

4.7 METRICS RELATED TO DEFECTS

1. **Derivative Metrics:** The team can use derivative metrics to discover the various software testing process problems and to take action that will improve testing accuracy.
2. **Defect Density:** Another crucial software testing measure, defect density aids the team in counting all the flaws discovered in a piece of

software during the course of its operation or development. The team can then determine whether the program is ready for release or whether additional testing is necessary by dividing the results by the size of that particular module. Software defect density is measured in terms of defects per thousand lines of code, or KLOC. The calculation is as follows: Defect Density = Defect Count/Size of the Release/Module

3. **Defect Leakage:** A crucial statistic that the testing team must track is defect leakage. Software testers utilize defect leakage to assess the effectiveness of the testing process prior to user acceptability testing (UAT) for the final product. Defect or bug leakage occurs when any flaws go unnoticed by the team and are discovered by the user. Defect Leakage is calculated as (total number of flaws discovered in UAT / total number of defects discovered prior to UAT) × 100.

4. **Defect Removal Efficiency (DRE):** The ability of the development team to eliminate various software problems prior to its release or implementation is measured by the DRE. DRE is calculated throughout and between test phases and is measured for each test type. It shows the effectiveness of the various defect removal techniques used by the test team. Additionally, it is a tacit evaluation of the software's performance and quality. As a result, the following is the formula for determining Defect Removal Efficiency: DRE = Number of defects resolved by the development team/ (Total number of defects at the moment of measurement)

5. **Defect Category:** During the software development life cycle (SDLC), this is a significant sort of metric that is assessed. The defect category metric provides information about the software's usability, performance, functionality, stability, reliability, and other quality characteristics. In short, the defect category is an attribute of the defects in relation to the quality attributes of the software product and is measured with the assistance of the following formula: Defect Category = Defects belonging to a particular category/ Total number of defects.

6. **Defect severity index:** The severity of a flaw as it relates to the operation or component of a software program under test is called the defect severity index. The defect severity index (DSI) provides information on the caliber of the product being tested and aids in evaluating the caliber of the testing team's efforts. The team can assess the degree of a negative impact on the software's performance and quality with the aid of this statistic as well. In order to calculate the defect severity index, apply the formula: Defect Severity Index (DSI) = Sum of (Defect * Severity Level) / Total number of defects

7. **Review Efficiency:** Reducing the number of software problems before delivery uses the review efficiency statistic. papers themselves as well as papers themselves may contain review errors. By using this statistic, one can cut down on the expense and labor involved in fixing or

resolving problems. Additionally, it confirms the effectiveness of the test case and lowers the likelihood of flaw leaks in later testing phases. The following formula is used to determine review efficiency: Total number of review defects / (Total number of review flaws plus Total number of testing defects) x 100 is the formula for review efficiency (RE).

4.8 USING DEFECTS FOR PROCESS IMPROVEMENT

Process improvement in Defect Management Process (DMP):

The Defect Management Process (DMP) prioritises discovered defects based on their severity before moving on to solve them. However, this does not imply that less serious flaws are not still present. Whether little or serious, a defect affects the system. procedure improvement is a procedure in which all flaws are viewed as severe and critical, necessitating their repair or resolution in all cases. Any kind of flaw that is fixed results in improved DMP processes. By preventing the occurrence of any kind of flaw that could have an impact on the system and lead to failure in the future, it also aids in reducing the number of defects.

Whether a fault affects the system more or less, it is still a serious problem when it occurs. However, occasionally developers and testers believe that defects with a low impact or severity are unimportant. Anything that leaves clients unsatisfied is a defect. This discontent may be the result of a flaw in the process, requirement, design, coding, testing, etc. The testing team must exert all of its efforts to assess and examine the procedure in order to determine the source of the defect. Further techniques and actions should be adopted to prevent similar flaws after root cause analysis. Organisations will be able to create high quality software products if they view defect as a process step rather than taking it for granted.

Goals of DMP:

- Senior Management must comprehend the severity of the fault and how it may affect the system in order to help the team and participate in the DMP.
- DMP should be carried out at each step and throughout the Software Development Life Cycle (SDLC) in order to improve the process.
- Each team should employ the DMP method because it greatly enhances workflow.
- Project objectives should be taken into consideration when choosing a development strategy and integrating it into the software development process because various projects have different aims.
- Processes should be evaluated frequently to spot errors early on, which will save money.

- The following actions should be taken to limit the likelihood of defect:
  - ✓Review test cases and test scenarios.
  - ✓Analyse and combine functional and non-functional requirements.
  - ✓Review technical specifications
  - ✓Baselines of the environment
- Using an automated project script, faults can be found at an early stage.

4.9 SUMMARY

You can evaluate quality before the product is produced thanks to software metrics, which offer a quantifiable technique to evaluate the quality of internal product attributes. Metrics give you the knowledge you need to produce effective requirements and design models, reliable code, and exhaustive tests. A software measure needs to be straightforward, calculable, compelling, consistent, and objective in order to be helpful in real-world settings. It should be independent of the programming language you're using and give you useful feedback.

Function, data, and behavior: the model's three component are the main metrics for the requirements model. Design metrics take into account concerns with architecture, component-level design, and interface design. Metrics for architectural design take into account the model's structural elements. By creating proximate measures for cohesion, coupling, and complexity, component-level design metrics give an indicator of module quality.

Software testing flaws, the defect management process, advantages, and disadvantages have all been observed.The Defect Management Process is crucial to software testing because, as we all know, faults must be tested in all software created code.Software flaws can be found and fixed as part of the defect management process. The entire defect management procedure will assist us in identifying the issue as early as possible and ensuring that a high-quality product is delivered.

Metrics and KPIs for software testing are significantly enhancing the software testing process. These play a significant part in the software development lifecycle, from validating the product's quality to guaranteeing the accuracy of the numerous tests carried out by the testers. Therefore, you can improve the efficiency and accuracy of your testing efforts and obtain superior quality by applying and putting these software testing metrics and performance indicators into practice.

4.10 LIST OF REFERENCES

1. Software Engineering for Students, A Programming Approach, Douglas Bell, 4th Edition, Pearson Education, 2005
2. Software Engineering – A Practitioners Approach, Roger S. Pressman, 5th Edition, Tata McGraw Hill, 2001

3. Quality Management, Donna C. S. Summers, 5th Edition, Prentice-Hall, 2010. 4. Total Quality Management, Dale H. Besterfield, 3rd Edition, Prentice Hall, 2003.

4.11 UNIT END EXERCISES

1. Explain the different types of Metrics.
2. Write a note on Complexity metrics.
3. Describe defect management.
4. Explain the Defect Management Process.
5. State the objectives, advantages and disadvantages of Defect Management Process.
6. State and explain various stages of Defect Management Process.
7. Describe and explain the metrics Related to Defects.
8. Write a note on Defect workflow and states