Software Testing and Quality Assurance

(Code : 410245(B) : Elective II)

Semester VII – Computer Engineering
(Savitribai Phule Pune University) (SPPU)

Strictly as per the New Credit System Syllabus (2015 Course)
Savitribai Phule Pune University w.e.f. academic year 2018-2019

Dr. K. S. Wagh
Associate Professor
AISSMS IOIT, Pune

Swapnil S. Shinde
Assistant Professor
Marathwada Mitra Mandal's College of Engineering,
Karvenagar, Pune

Tech-Max® Publications, Pune
Innovation Throughout
Engineering Division

PO310A

Software Testing and Quality Assurance

Dr. K. S. Wagh, Swapnil S. Shinde

Semester VII – Computer Engineering (Savitribai Phule Pune University)

First Edition : July 2018 **(As per New Syllabus)**

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

Printed at : Image Offset, Dugane Ind. Area, Survey No. 28/25, Dhayari, Near Pari Company, Pune - 41, Maharashtra State, India. E-mail : rahulshahimage@gmail.com

ISBN 978-93-88200-17-2

Published by

Tech-Max Publications

Head Office : B/5, First floor, Maniratna Complex, Taware Colony, Aranyeshwar Corner, Pune - 411 009. Maharashtra State, India.

Ph : 91-20-24225065, 91-20-24217965. Fax 020-24228978. Email : info@techmaxbooks.com, Website : www.techmaxbooks.com

[410245(B)] (FID : TP549) (Book Code : PO310A)

Preface

Dear Students,

We are extremely happy to present the book of **"Software Testing and Quality Assurance"** for you. We have divided the subject into small chapters so that the topics can be arranged and understood properly. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.

We are thankful to Late. Shri. Pradeep Lunawat and Shri. Sachin Shah for the encouragement and support that they have extended. We are also thankful to the staff members of Tech Max Publications and others for their efforts to make this book as good as it is. We have jointly made every possible efforts to eliminate all the errors in this book. However if you find any, please let us know, because that will help us to improve further.

We are also thankful to our family members and friends for their patience and encouragement.

- Authors

❑❑❑

Syllabus

Savitribai Phule Pune University

Fourth Year of Computer Engineering (2015 Course)

Course Code : 410245(B) : Elective - II

Software Testing and Quality Assurance

Teaching Scheme :	Credit	Examination Scheme :
TH : 03 Hours/Week	03	In-Sem (Paper) : 30 Marks
		End-Sem (Paper) : 70 Marks

Prerequisite Courses

310243- Software Engineering and Project Management, 310263- Software Modeling and Design

Companion Course : 410247-Laboratory Practice II

Course Objectives

- Introduce basic concepts of software testing.
- Understand white box, block box, object oriented, web based and cloud testing.
- Know in details automation testing and tools used for automation testing.
- Understand the importance of software quality and assurance software systems development.

Course Outcomes

On completion of the course, student will be able to -

- Describe fundamental concepts in software testing such as manual testing, automation testing and software quality assurance.
- Design and develop project test plan, design test cases, test data, and conduct test operations.
- Apply recent automation tool for various software testing for testing software.
- Apply different approaches of quality management, assurance, and quality standard to software system.
- Apply and analyze effectiveness Software Quality Tools.

Course Contents

Unit I - Introduction **(08 Hours)**

Introduction, historical perspective, Definition, Core Components, Quality View, Financial Aspect, Customers suppliers and process, Total Quality Management(TQM), Quality practices of TQM, Quality Management through- Statistical process Control, Cultural Changes, Continual Improvement cycle, quality in different areas, Benchmarking and metrics, Problem Solving Techniques, Problem Solving Software Tools.

Software Quality - Introduction, Constraints of Software product Quality assessment, Customer is a King, Quality and Productivity Relationship, Requirements of Product, Organization Culture, Characteristics of Software, Software Development Process, Types of Product, Criticality Definitions, Problematic areas of SDLC, Software Quality Management, Why Software has defects, Processes related to Software Quality, Quality Management System's Structure, Pillars of Quality Management System, Important aspects of quality management. **(Refer Chapter 1)**

Unit II - Test Planning and Management (08 Hours)

Review of Fundamentals of Software Testing, Testing during development life cycle, Requirement Traceability matrix, essentials, Workbench, Important Features of Testing Process, Misconceptions, Principles, salient and policy of Software testing, Test Strategy, Test Planning, Testing Process and number of defects found, Test team efficiency, Mutation testing, challenges, test team approach, Process problem faced, Cost aspect, establishing testing policy, methods, structured approach, categories of defect, Defect/ error/ mistake in software, Developing Test Strategy and Plan, Testing process, Attitude towards testing, approaches, challenges, Raising management awareness for testing, skills required by tester. **(Refer Chapter 2)**

Unit III - Software Test Automation (08 Hours)

What is Test Automation, Terms used in automation, Skills needed for automation, What to automate, scope of automation, Design and Architecture of automation, Generic requirement for Test Tool, Process Model for Automation, Selecting Test Tool, Automation for XP/Agile model, Challenges in Automation, Data-driven Testing. Automation Tools like JUnit, JMeter. **(Refer Chapter 3)**

Unit IV - Selenium Tool (08 Hours)

Introducing Selenium, Brief History of The Selenium Project, Selenium's Tool Suite, Selenium-IDE, Selenium RC, Selenium Webdriver, Selenium Grid, Test Design Considerations. **(Refer Chapter 4)**

Unit V - Quality Management (08 Hours)

Software Quality, Software Quality Dilemma, Achieving Software Quality, Software Quality Assurance. Elements of SQA, SQA Tasks, Goals, and Metrics, Formal Approaches to SQA, Statistical Software Quality Assurance, Six Sigma for Software Engineering, ISO 9000 Quality Standards, SQA Plan. **(Refer Chapter 5)**

Unit VI - Software Quality Tools (08 Hours)

Total Quality Management, Product Quality Metrics, In process Quality Metrics, Software maintenance, Ishikawa's 7 basic tools, Checklists, Pareto diagrams, Histogram, Run Charts, Scatter diagrams, Control chart, Cause Effect diagram. Defect Removal Effectiveness and Process Maturity Level. **(Refer Chapter 6)**

❑❑❑

UNIT I

Chapter 1 : Introduction 1-1 to 1-24

Syllabus : Introduction, Historical perspective, Definition, Core Components, Quality View, Financial Aspect, Customers suppliers and process, Total Quality Management(TQM), Quality practices of TQM, Quality Management through - Statistical process Control, Cultural Changes, Continual Improvement cycle, Quality in different areas, Benchmarking and metrics, Problem Solving Techniques, Problem Solving Software Tools.

Software Quality : Introduction, Constraints of Software product Quality assessment, Customer is a King, Quality and Productivity Relationship, Requirements of Product, Organization Culture, Characteristics of Software, Software Development Process, Types of Product, Criticality Definitions, Problematic areas of SDLC, Software Quality Management, Why Software has defects, Processes related to Software Quality, Quality Management System's Structure, Pillars of Quality Management System, Important aspects of quality management.

UNIT II

Chapter 2 : Test Planning and Management 2-1 to 2-30

Syllabus : Review of Fundamentals of Software Testing, Testing during development life cycle, Requirement Traceability matrix, essentials, Workbench, Important Features of Testing Process, Misconceptions, Principles, salient and policy of Software testing, Test Strategy, Test Planning, Testing Process and number of defects found, Test team efficiency, Mutation testing, challenges, test team approach, Process problem faced, Cost aspect, establishing testing policy, methods, structured approach, categories of defect, Defect/ error/ mistake in software, Developing Test Strategy and Plan, Testing process, Attitude towards testing, approaches, challenges, Raising management awareness for testing, skills required by tester.

UNIT III

Syllabus : What is Test Automation, Terms used in automation, Skills needed for automation, What to automate, scope of automation, Design and Architecture of automation, Generic requirement for Test Tool, Process Model for Automation, Selecting Test Tool, Automation for XP/Agile model, Challenges in Automation, Data-driven Testing. Automation Tools like JUnit, JMeter.

UNIT IV

Chapter 4 : Selenium Tool **4-1 to 4-14**

Syllabus : Introducing Selenium, Brief History of the Selenium Project, Selenium's Tool Suite, Selenium-IDE, Selenium RC, Selenium Webdriver, Selenium Grid, Test Design Considerations.

UNIT V

Chapter 5 : Quality Management **5-1 to 5-16**

Syllabus : Software Quality, Software Quality Dilemma, Achieving Software Quality, Software Quality Assurance. Elements of SQA, SQA Tasks, Goals, and Metrics, Formal Approaches to SQA, Statistical Software Quality Assurance, Six Sigma for Software Engineering, ISO 9000 Quality Standards, SQA Plan.

UNIT VI

Syllabus : Total Quality Management, Product Quality Metrics, In process Quality Metrics, Software maintenance, Ishikawa's 7 basic tools, Checklists, Pareto diagrams, Histogram, Run Charts, Scatter diagrams, Control chart, Cause Effect diagram. Defect Removal Effectiveness and Process Maturity Level.

❑❑❑

CHAPTER 1

Unit I

Introduction

Syllabus Topics

Introduction, Historical perspective, Definition, Core Components, Quality View, Financial Aspect, Customers suppliers and process, Total Quality Management(TQM), Quality practices of TQM, Quality Management through - Statistical process Control, Cultural Changes, Continual Improvement cycle, Quality in different areas, Benchmarking and metrics, Problem Solving Techniques, Problem Solving Software Tools.

Software Quality : Introduction, Constraints of Software product Quality assessment, Customer is a King, Quality and Productivity Relationship, Requirements of Product, Organization Culture, Characteristics of Software, Software Development Process, Types of Product, Criticality Definitions, Problematic areas of SDLC, Software Quality Management, Why Software has defects, Processes related to Software Quality, Quality Management System's Structure, Pillars of Quality Management System, Important aspects of quality management.

Syllabus Topic : Introduction

1.1 Introduction to Quality

- Human lifestyle is enhanced with the use quality products and services. Products take physical form whereas services can be termed as virtual form of product.
- Product (service) quality depends on two factors, one is skill of quality assessor and second, effectiveness of a process used to measure quality.
- Each and every product (service) is composed of various quality attributes.
- Due to immense competition, manufacturers must focus on keeping good identity of a product (service).
- Quality of product (service) depends on few characteristics like-satisfying customer's need, cost, features, functionalities, delivery schedule etc.

Syllabus Topic : Historical Perspective

1.2 Historical Perspective of Quality

- Concept of quality and quality improvement had emerged from the field of agriculture.
- In the 20th century, farmers were assisted for planning of the crops, rotation of cultivation, maintaining soil quality in order to maximize agricultural production.
- This work had conducted in Britain. Some researchers had contributed towards this and is described in below section.

1. **Walter Shewhart**

- Developed a quality improvement program with planned efforts.
- Applied concept to product manufacturing process to reduce cost to customer without decreasing profit.

2. **Dr. Edward Deming**

- Suggested Total Quality Management (TQM) program to improve quality methods and practices.
- Idea is demonstrated through continual improvement in Japan.

3. **Dr. Joseph Juran**

- Implemented quality improvement program through measurement processes.
- He had used different assessment and improvement tools for this.

- Concept of TQM was integrated by Japanese industry that leads to generation of quality products in sectors like, electronics, automotive etc.
- Product quality was established and continually improved in terms of cost, delivery schedule, performance, features etc.
- The Japanese products manufactured with proper organization and understanding of process used for their development. Use of different tools enabled them to monitor and understand processes.
- Japanese quality management program consists set of interrelated processes. This maintains same quality across large number of products.
- With continual process improvement program, root causes of defects are identified and removed easily.

Syllabus Topic : Quality Definition

1.3 Quality Definition

- It is defined as conformance to the specifications. Every product is having requirement specification, design specification etc.
- A quality product must conform to these specifications which ultimately satisfies the customer's needs. It must be built as per design stated.
- It must also satisfy customer's expectations related to cost, delivery time, support etc.
- Quality of a product is determined by its inherent attributes and characteristics. Term quality can be defined from various perceptions as follows.

1. Customer based definition

- A quality product must achieve customers satisfaction by meeting their needs and expectations.

2. Manufacturing based definition

- A quality product must be developed as per its requirement and design specification.
- Development methods must be selected wisely to produce required product with less or no defects.

3. Product based definition

- A quality product must possess set of attributes and characteristics that will help customers to satisfy their needs.
- These attributes will add great value to a product to make it distinguishable from other competing products.
- Such product will make customer to feel proud for having it.

4. Value based definition

- A product is a combination of cost and features (expected by customer).
- A customer must get returns for his investment after buying product.
- Cost of product depends on value found by customer in the product. Having more value for customer lead to better appreciation.

5. Transcendent Quality

- Many customers are not clearly knowing what quality is about a product they own.
- Instead they derive good characteristics from product that will delight them and gives proud feeling of ownership.

Syllabus Topic : Core Components

1.4 Core Components

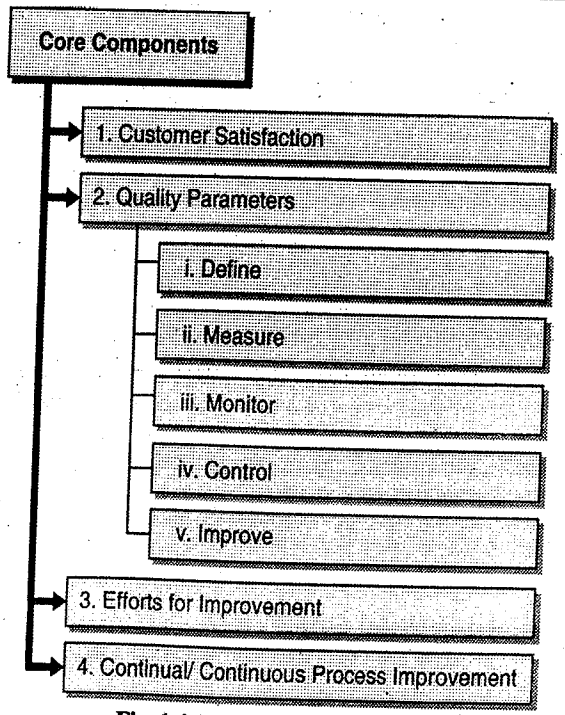


Fig. 1.4.1 : Core Components

➜ **1. Customer Satisfaction**

- Product quality is the perception of a customer made during its usage.
- Quality of a product will be determined based on level of satisfaction of a user.
- It is also decided by some attributes like, cost, delivery time etc.

➜ **2. Quality Parameters**

- Manufacturers must know the user expectations in order to achieve quality in product to be developed.
- By stating quality in measurable terms, it becomes easy for manufacturers to determine whether it has been achieved or not.
- Organizations must follow cycle of improvement (DMMCI) which has following components.

➜ **(i) Define**

- Product requirements in terms of attributes and characteristics must be stated in specification document. Acceptance criteria for a product must be defined.
- There must be a quantitative measurement of features, functionalities, and attributes of product.
- Supplier and customer, both must know what is required and what is not.

➜ **(ii) Measure**

- Quality of a product must measured in quantitative terms which can be defined as an attribute of quality.
- These quantities will act as an indicator for need of quality improvement.
- Lack in quality can be observed by comparing expected and delivered product.

➜ **(iii) Monitor**

- Organizations must establish monitoring mechanism to observe performance of processes used in development, testing, and delivery.
- Customer satisfaction must be ensured through this monitoring.
- Deficiencies in product must be handled by improving product and process.
- In such cases, organizations must set corrective and preventive action plans ready.

➜ **(iv) Control**

- Organizations must have functions like, quality control, validation, verification, and quality assurance.
- Product progress must be reviewed and controlled through these functions.

➜ **(v) Improve**

Continuous and continual improvement approaches must be followed by organizations to achieve maximum customer satisfaction, tackle the competition, and overcome the customer complaints.

➜ **3. Efforts for Improvement**

- Management of an organization should lead the quality improvement program by defining, vision, mission, objectives, goals, policies, strategies etc.
- Management must set an example by following these and encourage employees to adopt same.
- Management must define and approve different procedures, methods, standards etc.
- Deviations in system must be tracked that becomes the candidate for improvement.
- Quality plan developed by management always supports the improvement efforts and actions.

➜ **4. Continual/ Continuous Process Improvement**

- It is old practice where quality is improved with rework, testing, more inspection etc. Team was fixing defects when they are reported by customers.
- This has increased cost of inspection and rework, that in turn increases price for customer.
- Efforts for improving quality must be directed through continual/continuous improvement approach.
- Quality planning must be done to achieve target improvements. It must include processes, PDCA cycle (Plan-DO-Check-Act), and DMMCI cycle.
- Progress must be checked at each stage and required corrective actions must be stated.

Syllabus Topic : Quality View

1.5 Quality View

- A project or product is associated with different entities or people known as **stakeholders.**
- All stakeholders are interested in success, failure and betterment of project or product.
- Each one has different aspect towards quality as per their role.
- There are **total six major views** associated with product or project quality. These views are simply the expectations from project/product of an organization.
- If these views matches perfectly then no gap left in stakeholders expectations and will significantly help in improving performance.
- If there are differences in views then it will definitely affect the improvement efforts. Different views of quality are as follows.

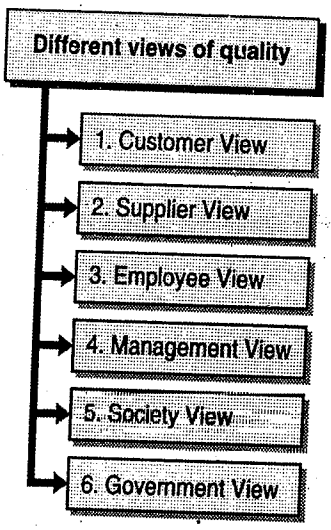


Fig. 1.5.1 : Different Views of quality

➜ **1. Customer View**

- Customer is considered as important stakeholder for any product or project.
- Customer is looking for a product that will benefit him in all regards like need, money, time, functionalities etc.
- So organizations must deliver product satisfying all requirements, within time, with all features included.

➜ **2. Supplier View**

- An important source of input for any product or project is supplier. External supplier can be hardware, software, other machines etc.
- Internal supplier can be people like trainers, administrators, contractual workers etc. Suppliers helps to make more business profit and expansion.

➜ **3. Employee View**

- Employees (permanent or temporary) are people working on organization project.
- Successful completion of projects will give employees more recognition, appraisal, satisfaction etc.

➜ **4. Management View**

- People responsible for managing operations of an organization are termed as management e.g. project management, senior management etc.
- Management is trying hard to succeed in their vision and mission. They need more business, profit, turnover, expansion etc.

➜ **5. Society View**

- Success or failure of project has effect over society. Successful projects increases economics of society to great extent.
- Successful organizations increases employment opportunities for neddy and qualified people.

➜ **6. Government View**

- It can be local or state or central and may be affected due to success or failures of organizations.
- There will be a noticeable improvement in nation's progress If people gets good wealth and employment.
- Government can gain more taxes, currency, and export benefits.

Syllabus Topic : Financial Aspect

1.6 Financial Aspects of Quality

- In early days, product price was the only indicator of quality. High price product was considered as of best quality.
- Selling price is decided by addition of cost of manufacturing, cost of quality and profit.
- In order to maintain profit, manufactures started reducing cost of manufacturing without compromising quality. In the next section these costs are discussed in detail.

1.6.1 Cost of Manufacturing

- It is the cost associated with requirement analysis, design, development, testing etc.
- Resources like licenses, people, and material also involved in cost of manufacturing.
- Time and processes related to project are associated with amount of efforts.
- Technological changes can make improvements in processes that leads to increased productivity.

1.6.2 Cost of Quality

- It is the cost associated with efforts made for improvement and maintenance of quality. It has relationship with cost of manufacturing.
- Generally efforts are made towards defect prevention, product appraisal, defect removal, product acceptance, regression testing etc.

1. **Cost of Prevention (Green Money)**

- It is associated with defined processes, standards, development, testing, training, guidelines, checklists, formats, templates, process models etc.
- Organization is expecting returns on investment.
- Single part of green money can reduce 10 parts of blue money and 100 parts of red money.

2. **Cost of Appraisal (Blue Money)**

- It is associated with cost incurred on first time reviews and testing.
- Returns on investment is not possible but it helps to identify process capabilities and problems.
- One part of blue money can reduce 10 parts of red money.

3. **Cost of Failure (Red Money)**

- It starts incurring with defect detection and fixing at any development stage.
- Rework, retesting, regression testing, sorting etc. are typical examples of it.
- It may result in customer dissatisfaction, loss of faith, business loss etc.

Syllabus Topic : Customers, Suppliers and Processes

1.7 Customers, Suppliers and Processes

- Every organization is managing its overall supply chain. Internal and external suppliers are already discussed in previous section.
- These entities are providing required inputs to the organization. Supplier of one organization can be the customer for another organization.
- Internal functions and projects are termed as internal customers. It is supported and serviced by other functions or projects.
- During supply chain, it is necessary that each function shall understand its customers, suppliers and their needs in order to fulfill requirements.
- This forms strong basis for TQM. If internal customers are satisfied then it will have positive effect on external customers.
- People that are external to an organization are termed as external customers.
- This entity is using product and services offered by organizations by paying them.
- By focusing on satisfaction of external customers, product quality can be improved.

Syllabus Topic : Total Quality Management (TQM)

1.8 Total Quality Management (TQM)

- TQM approach describes internal, external customers and internal, external suppliers for each process, project and for entire organization.
- Process and function of organization can be broken down into different components which may act as customer or supplier to each other during process workflow.
- TQM approach is based on quality principles and is applicable to all aspects of organization (e.g. business processes).
- Fig. 1.8.1 represents supply chain relationship between customer and supplier

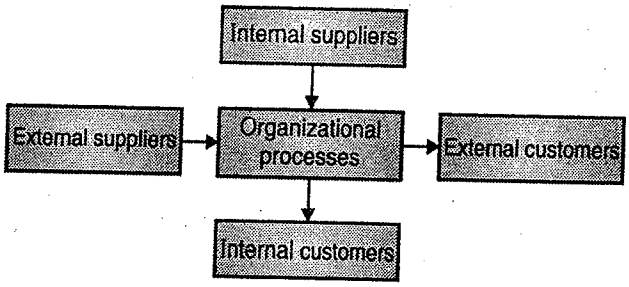


Fig. 1.8.1 : Supply chain relationship

- Dr. Edward Deming implemented QMS based on TQM and Continual improvement in Japanese environment.
- It resulted into repetitive, cost effective processes aiming at achieving customer satisfaction.
- Dr. Deming proposed 14 quality management principles that are widely used by quality practitioners.

Syllabus Topic : Quality Practices of TQM

1.9 Quality Practices of TQM

1. Create constancy of purpose toward improvement of products and services. The aim behind this is to become competitive, to stay in business, and to provide jobs. Organizations must Plan for long term quality and profits.
2. Adopt the new philosophy as we are in altogether new economic age. It is unacceptable to live with commonly accepted levels of delays, rejections, mistakes, defective materials and bad workmanship. Transformation of Western management style into TQM is necessary to drive business and industry towards improvement.
3. Cease dependence on inspection to achieve quality. Eliminate the need for inspection of entire product by employing quality checks from beginning of development. Inspections will find only lack in quality and will not help to improve quality. It is recommended to use statistical measure to control and improve development process.
4. Stop the practice of awarding business on the basis of price tag. Focus must be kept on minimizing total cost instead of initial cost. Quality is highly dependent on consistency so if you have less variations in input, obviously there will be less variations in output. A single supplier can be contacted for any one item, on a loyal, trustworthy, good, long-term relationship. Further, to ensure that suppliers meet your quality standards, quality statistics can be used.
5. Constant improvement in the processes related to product and services must be done. This will improve quality and productivity, which in turn leads to cost reduction.
6. Use training and development programs for employees on the job. This will create strong foundation of knowledge and skills required at work.
7. Institute leadership at workplace to help people to do a better job. Main aim is to provide required support and resources to people and machines.
8. Eliminate fear so that everyone can feel free and work effectively for the organization. Make use of open and honest communication to express ideas or concerns. This will motivate employees, increases respect and interest in doing work.
9. Break down barriers between departments and staff areas. Focus on collaboration and consensus is very important. Build shared vision approach by forming cross-functional teams.
10. Eliminate slogans, exhortations, and targets for the workforce asking for zero defects and new levels of productivity. Such exhortations only create adversarial relationships, as the bulk of the causes of low quality and low productivity belong to

the system and thus lie beyond the power of the workforce. Eliminate work standards (quotas) on the factory floor. Substitute leadership. Eliminate management by objective. Eliminate management by numbers, numerical goals. Substitute leadership.

11. Eliminate work standards that uses numerical quotas for the workforce and numerical goals for management. Substitute it with motivation and helpful leadership that will drive employees to work towards the long term goal and vision of the organization. According to Deming, production targets are always encourage high output and low quality. With proper support and resources, production levels and quality can be high and achievable. Processes must be measured instead of the people behind the process.
12. Remove the barriers that take away pride of workmanship. Everyone should be allowed to take pride in their work without being rated or compared. Give similar treatment to each and every worker, and don't make them to compete with other workers for monetary or other kind of rewards. The quality system will automatically raise the level of everyone's work to an equally high level.
13. Implement education, training and self-improvement program for everyone. Improve the current skills of workers and encourage them to learn new skills to prepare for future changes and challenges. Building required skills will make your workforce more adaptable to change, and better able to find and achieve improvements.
14. Clearly define top management's commitment towards improvement in quality and productivity. Employees must perceive the top management commitment for quality and productivity. Employee support is not enough rather, they should take action in order to accomplish the transformation.

Syllabus Topic : Quality Management through - Statistical Process Control

1.10 Quality Management through - Statistical Process Control

- Statistical quality control proposed by **Dr. Juran** through DMMCI improvement cycle approach.
- Quality management must be done based on measurements and by understanding interrelationship between customers, suppliers, and processes of different stages (development, testing etc.).
- There are three factors of this approach and are as follows.

1. Quality planning at all levels

Quality is a result of planned efforts taken by all involved entities. Quality planning can be done at following two levels.

(i) **At Organization level :** Planning must be done at this level in the form of policy document based on vision, mission etc. Planning activities must identify current and future customers, and their needs. This must be quantified so that actions can be planned and progress can be measured. Measurements will highlight level of quality.

(ii) **At Unit level :** Planning must be done by people. Quality plans must be inline with policies and strategies and must ensured for consistency.

2. Quality control

It examines current product/process across various levels of standard. Process is made defect-free to improve its capability and quality. In case of deviation in process, it is measured against quality plan and actions are taken to minimize it.

3. Quality improvement

It is used for continuous improvement of process quality. The quality attributes of a process are measured and improvement areas are identified.

Syllabus Topic : Cultural Changes

1.11 Quality Management through - Cultural Changes

Quality improvement through cultural changes approach is given by Philip Crosby. These cultural changes are driven by management of an organization.

It involves

- By doing capability measurement identify improvement areas in process.

- Give priority to certain areas based on available resources, cost benefit analysis, pareto analysis, efforts required etc.
- Deploying teams under management directions for improving development, testing, maintenance processes can bring cultural changes.
- Goals and targets can be set to check process progress and improvements.
- Customer satisfaction levels and competition are the main sources for deciding goals and targets.
- Giving recognition to the teams will increase morale and establishes positive and competitive environment in the organization.
- Repeating quality improvement cycles can help in maintaining improvement status.

Syllabus Topic : Continual Improvement Cycle

1.12 Continual Improvement Cycle

Plan-Do-Check-Act (PDCA) activities forms a cycle of quality improvement in TQM approach. Its stages are as follows.

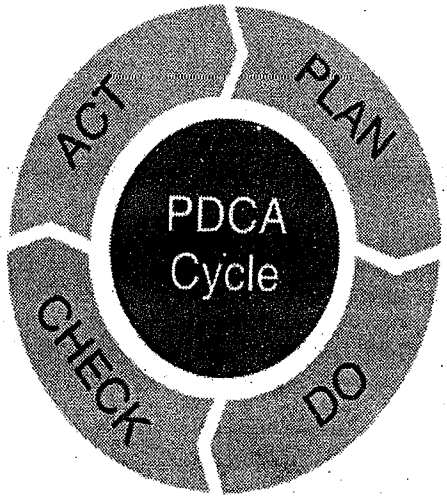


Fig. 1.12.1 : PDCA Cycle

1. **Plan :** Improvements must be planned using vision and mission of an organization. Synchronize both quality plans i.e. organization level and unit level with each other. Define expected outcomes in quantifiable terms and plan actions for deviations. Use baseline studies to understand current progress.
2. **Do :** Execution of set plan is important to achieve decided outcomes. It needs resources like training, hardware, software etc.
3. **Check :** Actual outcomes must be checked against planned one. It should be in measurable terms for easy analysis. This decides the direction of progress.
4. **Act :** Comparison between planned and actual outcomes will give degree of variation. The observed variations are corrected by taking actions (change in plan).

Syllabus Topic : Quality in Different Areas

1.13 Quality in Different Areas

- In this section, some product/service domains and its customer expectations (related to quality) are listed.
- Quality expectations are depend on service domain, customers type, features etc.
- Different domains has their different quality factors. Some of the basic quality parameters are in many areas are as follows :
  - Product price
  - Delivery time of product
  - Customer satisfaction
  - Number of defects
  - Support services

Table 1.13.1 shows list of domains and respective customer expectations.

Table 1.13.1

Sr. No.	Category	Expectations
1	Airlines	On-time, comfortable, low-cost service
2	Health Care	Correct diagnosis, minimum wait time, lower cost, security
3	Food	Good product, fast delivery, good environment
4	Postal	Fast delivery, correct delivery, cost containment
5	Academic	Proper preparation for future, on-time knowledge delivery
6	Consumer Products	Properly made, defect-free, cost effective

Sr. No.	Category	Expectations
7	Insurance	Payoff on time, reasonable cost
8	Military	Rapid deployment, decreased wages
9	Automotive	Defect-free, Mileage, Spare Parts availability, fast service
10	Communications	Clearer, faster, cheaper service

Syllabus Topic : Benchmarking and Metrics

1.14 Benchmarking and Metrics

(1) Benchmarking

- It is an useful concept of QFD (Quality Function Deployment) and defines measurable variables (scales).
- It is used to create qualitative and quantitative metrics that measures product quality against various scales of a benchmark.
- Examples of benchmarking variables are - cost, time, defect count, satisfaction level, functionalities etc.

(2) Metrics

- It represents relative measurement of product parameters and processes used to produce it.
- Metrics collects information about process variations, product capabilities, product attributes etc. Metrics are defined from planning document, benchmarks etc.

Syllabus Topic : Problem Solving Techniques

1.15 Problem Solving Techniques

- Problems associated with development and improvement of processes are handled with metrics based methods and techniques.
- Problem solving is done by both quantitative and qualitative method.
- In qualitative problem solving method, problem solutions are understood using quality index such as low, medium high.
- This indicates improvement or deterioration in the parameter. It is suitable for low maturity organization where problem size is broad and is divided into number of stages.
- In quantitative problem solving methods, exact measurement (numerical value) of parameter is used.
- It is used in high maturity organization where improvements are done on repetitive basis. It follows DMMCI cycle and measurements are required to be very accurate.
- Process is well understood by applying statistical technique on quantitative data. This data enables easy visualization and analysis of a process.
- Variations in a process can be identified easily and actions may be initiated to reduce it.

Syllabus Topic : Problem Solving Software Tools

1.16 Problem Solving Software Tools

- Organizations need to invest reasonably large amount to buy analytical tools, data management softwares etc.
- Tools are helpful in understanding the problem and suggests solution based on data inputs.
- Tools can be hardware, software, physical or logical in nature. Quality tools are used by projects teams to solve problems faced by them.
- Techniques is concerned with process used for measurement, analysis, and decision making. Techniques are independent of tools but supports use of tool.

☞ Advantages of using Tools

1. Tools are highly accurate and fast to make calculations and transactions. Calculations acts as input for decision making thus need to be accurate.
2. Decisions made by tools are with negligible variations. Some tools can be used on fixed range whereas some can learn the things and then used.
3. Tools reduces manual work and delivers results faster and accurate.
4. It can be integrated with other systems for complex problem solving.

☞ Disadvantages of using Tools

1. Tools need to learned by sparing significant amount time. Some needs training that incurs time as well as cost.

2. Softwares and hardware can suffer from defects which may affect decision in future.
3. Tools only suggest solutions but humans make decisions. Some tools can take decisions to a limited range only.

Syllabus Topic : Software Quality - Introduction

1.17 Software Quality

- **Quality of a software is defined as conformance to the specifications.** All functional and non-functional requirements are documented. Some requirements are not stated by customer but are necessary for building product, so such requirements are assumed by developer.
- Generally this document is approved by customer and then by developers.
- Quality of deliverables is totally depend on documented and approved software requirement specification document. Again, there are many constraints on this document.

Syllabus Topic : Constraints of Software Product Quality Assessment

1.18 Constraints of Software Product Quality Assessment

- Business analyst or system analyst are responsible for creating product requirement specification.
- Testing personal cannot have direct connect with customer and get to know information through requirement document, feedbacks, queries etc.
- Such scenarios are creating some constraints while assessing product quality which are as follows :
  - As compared to other engineered product, a software is not produced in a physical form. Thus human senses (like touching, hearing, seeing) and measuring instruments are of limited capability.
  - There is huge communication gap between customers and development-testing team.
  - Software is considered as unique product, but there exist similarities among many products. Properties like, requirements, design, architecture, coding, testing, reusability may highlight significant differences.
  - Software can not be tested fully. Exhaustive testing is concerned with cost.

Syllabus Topic : Customer is a King

1.19 Customer is a King

- Every SDLC process is checked against customer requirements for its conformation. Customer is treated as very important entity during product development.
- Each organization is trying to improve and exceed their customer's satisfaction level.
- Customer is always informed about whatever extra things provided to him in order to accept or reject those.
- Satisfied customer is considered as an achievement by all organizations. It leads to have repeat orders, trust building, more customer references etc.
- Following are some of the factors that determine success criteria of an organization.
  - Internal customer are those, working in different groups or function or department. Persons from other department or function acts as internal supplier for them.
  - TQM approach drives satisfaction of internal entities (customer and supplier). End users are considered as external customers.
  - End users need is documented in requirement specification document by development team. External suppliers are entities outside of organization.

Syllabus Topic : Quality and Productivity Relationship

1.20 Quality and Productivity Relationship

- Many organizations or people are thinking that more testing, inspection, rework gives a good quality product.
- More inspection would find more defects and a good defect free product can be delivered. It implies more cost, time and efforts along with less profit.

- Product quality can be improved by improving quality in production and testing process.
- This will automatically reduces inspection, rework, testing, wastages etc and improves productivity and in turn customer satisfaction.
- Most of the times, customers are complaining about problems related to product and services offered by an organization.
- These problems are the outcome of faulty processes used during development that gives rise to number of defects.
- Improved quality can reduce cost of development, cost of quality, selling price thus increases profit margins.
- Employee is treated as an important entity during the phase of quality improvement and performance improvement.
- Employees can detect and correct deviations in development process. This is because they are closely working with processes.
- Contribution of both, management and employee forms strong basis for organization improvement. Lack of either will result in customer dissatisfaction, less profit, loss of trust.
- Less or excessive amount of communication is considered to be a major problem in the businesses. Miscommunication leads to wrong interpretation which in turn can leads to user's gap and producer's gap.
- With management guidance and support, employees are working and converts organization into performing teams.
- With daily contribution, a new, improved working process can be innovated. Instead of waiting for inventions for better product, processes can be improved to achieve long term goals.
- Smart work will help the employees to identify deviations in development process thus can be eliminated easily.
- Research and Development department are carrying out inventions in many organizations.
- It is directed to develop new technologies and techniques in the process approach of development and testing.
- Six sigma approach suggests refinement, redesign in the existing processes.

Syllabus Topic : Requirements of Product

1.21 Requirements of Product

- Every product offered to customer must satisfy all his requirements and needs.
- To achieve this, all the phases of SDLC are driven according to requirements and needs.
- Due to fierce competition in the business world, product requirements are difficult to categorize.
- In such cases, organizations are focusing on the customers and decides priorities of requirements. Following are the different categories of requirements.

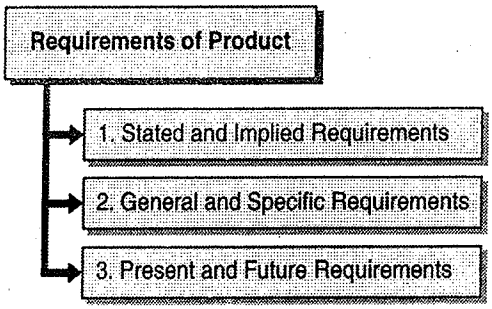


Fig. 1.21.1 : Requirements of Product

➔ **1. Stated and Implied Requirements**

- Stated requirements are documented in SRS (Software Requirement Specification) document.
- Business analysts and customer specifies the functional and non-functional requirements in the SRS document.
- Development and testing teams must be able to understand stated requirements.
- There are certain requirements which may not be documented but are considered in product. e.g. Readable font size, no spelling mistakes etc.
- Business analysts are supposed to convert implied requirements in stated one.

➔ **2. General and Specific Requirements**

- General requirements are accepted for a type of product and group of users, whereas some requirements are very specific for a product in development.

- General requirements are considered as implied one and specific requirements are considered as stated. General requirements can be, Multiplication should be correct, Easy to use Interface etc.
- Specific requirements can be, Six digit precision in float calculations, Authentication followed by notification etc.

➔ **3. Present and Future Requirements**

- Present requirements are considered for an application use in current circumstances. Future requirements are considered after some time span.
- Both requirements are need to be finalized by customer and business analysts. Development team need to identify future needs of customer by doing research.
- For example, current requirement of a banking software is of 1000 online accounts, but within three years it can grow to 10000.

☞ **Requirement Categories Based on Priority**

The requirement categories based on priority of their implementation from user's perspective.

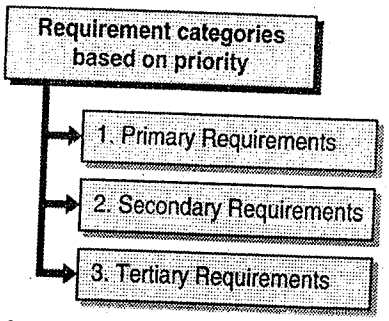


Fig. 1.21.2 : Requirement Categories Based on Priority

➔ **1. Primary Requirements**

- It includes "Must" and "Must not" type of requirements for which customer is paying. These are essential requirements and are denoted by P1 indicating high priority.
- Value of a product depends on accomplishment of these requirements or else product will be rejected due to dissatisfaction.
- It covers "Must not" requirements also, that are not present in product.

➔ **2. Secondary Requirements**

- It includes "Should be" and "Should not be" type of requirements. It adds some value to the product and are appreciated by customers.
- Customer may pay little extra for having these type of requirements in product. Its presence may delight customer and its absence may disappoint little.
- It is denoted by P2 priority and covers "Should not be" requirements also.

➔ **3. Tertiary Requirements**

- It includes "Could be" and "Could not be" type of requirements. It is not adding much value in terms of price paid by customer.
- If two products are same then "could be" requirements can give competitive advantage and receives customer appreciation.
- It helps to provide identity to a product and are denoted by P3 priority. It is lowest priority requirements and also covers "Could not be" requirements.

Syllabus Topic : Organization Culture

1.22 Organization Culture

- Management perspective, employee involvement and reason of existence are the three main factors behind organization culture.
- Quality improvement programs are depending upon organization's ability to bring cultural change.
- In the earlier section, we have seen the quality improvement for cultural change that is prescribed by researcher Philip Crosby.
- Quality culture is simply views of organization towards their stakeholders, employees, customers, suppliers etc.
- The two types of organizations can be formed based on the quality culture they follow and are shown in Table 1.22.1.

Table 1.22.1 : Difference between Q - Organization and 'q' Organization

Sr. No.	Q - Organizations	q - Organizations
1.	More concerned about quality	Less concerned about quality
2.	Listen to customers and understand their requirements.	Assumes customer requirements.

Sr. No.	Q - Organizations	q - Organizations
3.	Focus on cost of quality to reduce cost of failure in turn reduces overall price.	Focus on cost of poor quality because they believe more testing gives quality product.
4.	Way of working - First time right product	Way of working - rework, inspection, scrap, sorting
5.	Focusing on : continual/continuous process improvement.	Focusing on : finding and fixing defects.
6.	Organization takes ownership of processes and all level defects.	Organization assigns responsibility of defect handling to someone.
7.	Organization shows leadership and commitment towards quality and customer satisfaction	Organization assigns responsibility of quality to others.

- As organization progresses from 'q' to 'Q', there is a cultural change in perspectives of management and employee towards quality and customer satisfaction.
- In the initial stages (higher side of 'q'), product undergoes inspection, rework, sorting etc to ensure defect free product delivery to customer.
- In the final stages, the 'Q' organizations works on defect prevention through process improvements.

Fig. 1.22.1 : Focus shift in Quality

Quality control

- Organizations at higher end of 'q' thinks that customer satisfaction can be achieved by delivering defect free product to them.
- Quality control approach based on defect finding and fixing through heavy inspection, testing, rework, scrap, sorting activities.
- There are big testing teams, large appraisal cost, defect fixing cost, regression testing etc.

Quality assurance

- Next level of improvement is shift from quality control to quality assurance.
- It focuses on first time right product philosophy instead of fixing and testing activities.
- Major investment is done for defining policies, methods, process for various functions of organization.
- Corrective and preventive actions are planned based on metrics program.
- Defects are considered as process failure and corrective actions are deployed to optimize processes.

Quality management

- This approach is implemented by highly mature organizations. The processes are optimized on continuous basis and are defect free thus delivers a quality product to their customers.
- Organizations are always working on continuous improvement plans and defect prevention techniques.
- They define various processes, methods, techniques, training programs for future technologies and process improvements.
- Management starts concentrating on achieving desired output and organizational objectives through the activities like planning, organizing, directing, coordinating, staffing, controlling, coaching etc.

Syllabus Topic : Characteristics of Software

1.23 Characteristics of Software

1. Software is unique in nature with respect to characteristics, performance, capabilities etc even though they are satisfying similar needs of customer.
2. Softwares are virtual entities and are executable thus it can not be sensed with general inspection/testing methods.
3. Softwares cannot be measured with general measuring instruments. It needs testing and it is impossible to have exhaustive testing on it.
4. Software executes in a same way when it is executed every time. Algorithms and conditions comprising a software can have multiple combinations. Checking and testing of all such combinations is impractical

Syllabus Topic : Software Development Process

1.24 Software Development Process

It is also referred as SDLC that defines how the software is being developed. Following are the development approaches.

1. Waterfall model

- It is termed as classical view of software development and forms foundation for any development activity. It is one of the simplest model but not feasible to work every time.
- Different variants are available such as, modified waterfall model, iterative waterfall model etc. Many other development models are basically depend upon waterfall model.
- Customer requirements are converted into low level and high level design. Code is implemented for the design and tested as per test plan.
- Tested code deployed at customer site and it is maintained in last phase. This model provides short route for development activities and is highly efficient and productive in nature.
- There are some limitations of waterfall model. It is more useful when projects are having fixed schedule and price.
- This model is not having any provision for feedback (improvements) because it assumes that requirements are stable and no error will occur during entire SDLC.
- This model does not involve any kind of rework.

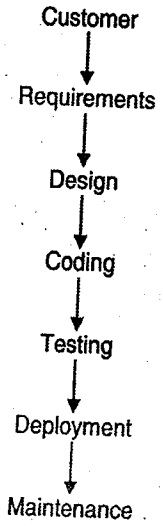


Fig. 1.24.1 : Waterfall Model

2. Iterative model

- Unlike waterfall model, this model does not assume that customer will provide requirements in one go and those will be stable one.
- Changes are assumed in any phase of SDLC. This is accommodated by introducing feedback loop that makes it different from waterfall model.
- There are some limitations of iterative model. It consists of many number of cycles of waterfall model. It is not fit for projects having fixed price and schedule.
- Due to many iterations, product design and architecture becomes more fragile.

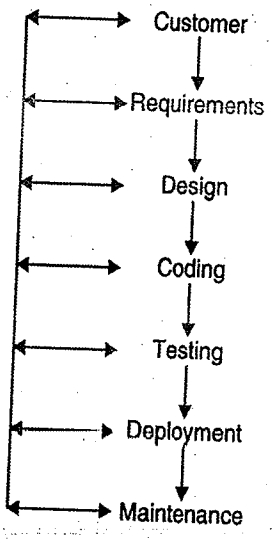


Fig. 1.24.2 : Iterative Model

3. Incremental model

- It is generally used to develop large systems consists of many subsystems as their component. These subsystems can be developed using waterfall or iterative model.
- Later, these developed subsystems can be connected directly or indirectly (using interconnection application).
- Incremental model develops a system and customer starts using it. Meanwhile second system is developed and integrated with the first one and so on. This model does not require requirement at the start.
- There are some limitations of incremental model. Direct connectivity could lose the flexibility of application.
- System integration can be a big challenge in this approach. Complex integration of subsystems may cost change in system architecture and the loss of flexibility.

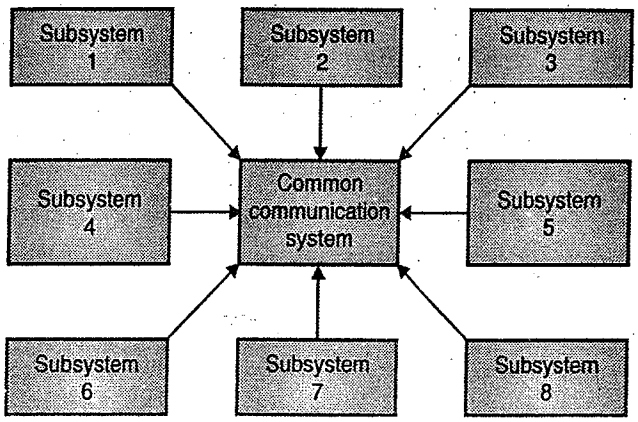


Fig. 1.24.3 : Incremental Model

- Increment requires regression testing to verify correctness of integration.

4. Spiral model

- In this approach, requirements are received in multiple iterations and according to it the development is carried out. Systems with incrementing size are built using spiral model e.g. ERP softwares,
- Banking system etc. Initially some functionality is created and given to customer.
- After using it, customer adds few more requirements into it. In this way software is developed in a spiral way. There are some limitations of spiral model.
- It needs refactoring and change in approach when initial system is non-usable. It also requires multiple regression testing cycles after addition of new functionality.

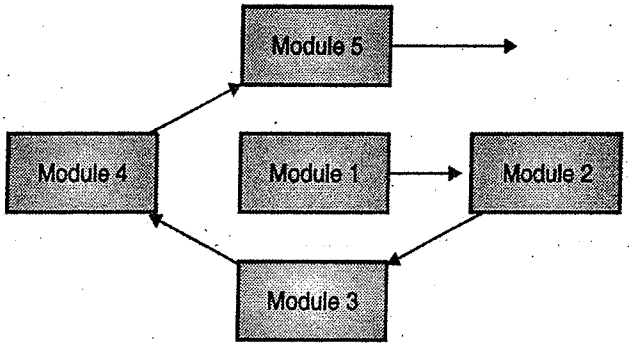


Fig. 1.24.4 : Spiral Model

5. Prototyping model

- This model contains top-bottom reverse integration approach. Many times customers requirements are not clearly understood then prototyping approach can be used there.
- At the start a prototype of system is created and given to customer.
- This will help them to understand their expected product. It is helpful for development team as they can understand system look and feel.
- Upon finalizing the requirements system is built and product is delivered. There are some limitations of prototyping model. Model of a system is created not the actual product.
- Due to this customer may pressurize development team to deliver it immediately.

6. Rapid application development model

- It develops usable software at a fast speed where user still understands development and application under process. It works similar to spiral model.
- Developers starts with very less number of requirements and creates design, code it, test it and delivered to customer.
- Once software is delivered, customer can understand his expectations and development in a more clearer way. He can add more or refine earlier requirements.
- In each iteration this is followed by development. Major constraints in rapid application development model are change in approach and refactoring.
- Each iteration requires multiple regression testing cycles.

7. Agile development model

- The model is dynamic in nature and has adaptability to user environment and continuous product integration.
- The importance is given to deliver a working product through customer collaboration, instead of fulfilling requirements from SRS document.
- It gives complete freedom to customer to add their requirements at any phase of SDLC and developers must accept those.
- Agile development consists of methodologies like, scrum, extreme programming, test driven development etc.

8. Maintenance development model

- Maintenance of a software is one of the major costly phase. Defects in a system can create long term problems.

- New technologies are introduced that offer better performance, services, other option in a cost effective way.
- Many times new functionalities are added as a part of business need.
- Following are the four main categories of software maintenance activities.

(i) **Bug fixing :** Bugs or defects are fixed by doing proper analysis of it. Sometimes fixing old bugs may introduce new one in the system.

(ii) **Enhancement :** New functionalities are added as per customer requirements or business need.

(iii) **Porting :** Software functionalities are ported from older technology platform to newer one. Features of older platform are expected to be present in new platform.

(iv) **Re-engineering :** Due to change in business environment there is need to change logical parts and algorithms used in development.

Syllabus Topic : Types of Product

1.25 Types of Software Product

Software products can be categorized based on their criticality i.e. their importance to customer. Following are the four main categories.

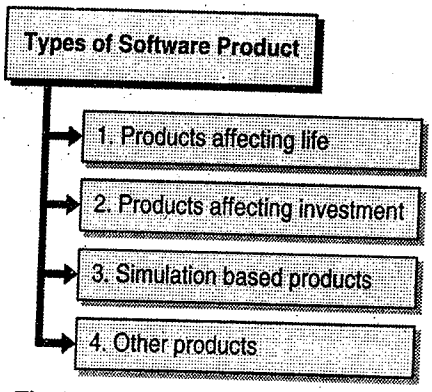


Fig. 1.25.1 : Types of Software Product

➜ **1. Products affecting life**

- Products from this category are the most critical product since they can directly or indirectly affect the human life. These products are having regulatory and safety requirements.
- It takes normal customer requirement, precise quality requirement and undergoes critical testing process since failure can lead to death or disability.
- Such products are again having five subcategories which are as follows :

(i) Most critical product, where failure resulting into death of a person.

(ii) Second level critical product, where failure resulting into permanent disability.

(iii) Product failure resulting into temporary disability

(iv) Product failure resulting into minor injury.

(v) All other products that do not affect health.

➜ **2. Products affecting investment**

- Products from this category are ranked second in the list criticality. They can have effect over investment made.
- It takes many regulatory and statutory requirements along with large testing efforts (but less than the first category products). e.g. e-commerce softwares.
- Quality factors for such products are security, accuracy, confidentiality etc.

➜ **3. Simulation based products**

- Products from this category are difficult to test in real world hence are tested with simulators.
- These products ranked third in the list of criticality. e.g. products from space research, aeronautics etc.
- They need large amount of testing but less than above two categories.

➜ **4. Other products**

All products other than above three are put in this category.

Syllabus Topic : Criticality Definitions

1.26 Criticality Definitions

Following are some more ways of classifying criticality of a product. It is having direct relationship with business and failure may lead to huge lose to organization or customer.

1. This classification talks about dependency of business on system i.e. complete dependency or minimal dependency.

(i) Product whose failure leads to business destruction are termed as most critical from business perspective. In such cases no fallback (backup) arrangements are possible.

(ii) Failure of product affects business partially since fallback (backup) arrangements are possible. Effect on business, services and profits is temporary and can be restored with some efforts.

(iii) Failure of product does not have any effect on business since there are other methods to get same outcomes. Re-arrangements are easily available without any disturbance.

2. This classification talks about products operating environment.

(i) Environments like aeronautics, space research are very complex and considered as very critical. Product failure can cause major problem since operating environment is not an easy one.

(ii) Environments like banking system are less complex than the first type. In case of failure huge computations can be affected but are recoverable.

(iii) Products under this type are operating in very simple environment. Failure of it will not have severe effects and can be recovered quickly or some alternative arrangements are done.

3. This classification talks about complexity of a system on the basis of development capabilities required.

(i) User data inputted through form based application are stored in database. When required it can be retrieved easily. No complex manipulation is performed on data.

(ii) Applications based on algorithms performs large number of operations and output is generated. Further decision are taken based on these outputs. Design, development and testing of such system is complex due to involvement of mathematical models.

(iii) AI based systems (Artificial Intelligence) are very complex. They learn the things (after storing into it) and use it when required.

Syllabus Topic : Problematic Areas of SDLC

1.27 Problematic Areas of SDLC

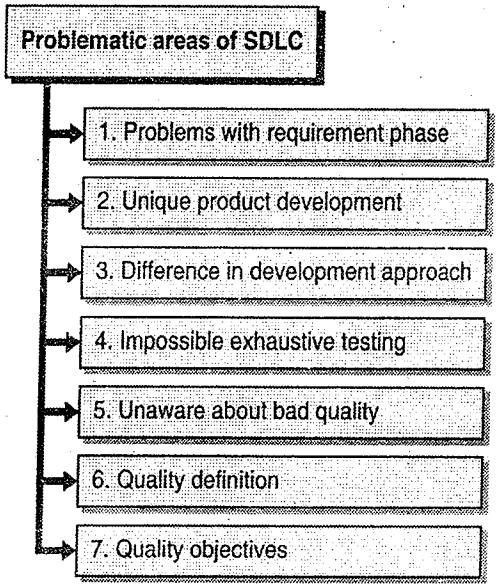


Fig. 1.27.1 : Problematic Areas of SDLC

➜ **1. Problems with requirement phase**

Requirement collection is an important phase in SDLC and having highest probability of introducing defects in the system. Following are some of the problems.

(i) **Communication**

It is considered as major problem in forming requirement statement. Many times requirements are not communicated in an easy way.

(a) Technical requirements are about platform, language, database, OS etc. It also consist configuration of various technical entities. Generally development team deals with these requirements.

(b) Economics of software is based on its technical and system requirements. Generally development team and customer are dealing with this type of requirement. Customer must understand different approaches and select one out of it. Development organization must help customer in choosing right approach by sharing its experience.

(c) Software product may be associated with different statutory and regulatory requirements. There may be some rules and regulations applicable to product and those must be understood by development team.

(d) Operational requirements are combination of different functional and non-functional requirements. These are defined by user/customer based on their business needs. It defines what the software must do and must not do.

(e) Security requirements are combination of physical and logical security requirements. Customer and development team are working with these. It consists requirements for backup, restoration, configuration, access control, hardware etc. Customer may specify requirements for privileges encryption, password protection etc.

(ii) Changing nature

- Dynamic nature of requirements resulting many complaints from development teams.
- Changing requirements may confuse the developers. After showing built product to customer, many new ideas are suggested and some of them directly have effect on cost, time, and efforts.
- Sometimes changes suggested may have effect on design, architecture, and methodology also. The time gap between requirement submission and product delivery plays very essential role.
- Approaches like rapid application development, top-down, joint application development can be used to accommodate changing requirements.

➜ **2. Unique product development**

- In software field no two application are same. Implementation for same problem, done by two different developers differ from each other.
- Even same solution developed by same developer at two different instances may not be similar. Software produced in such instances may be considered unique.
- During maintenance, designers facing difficulties in understanding original design whereas developers faces difficulties in reading the code.

➜ **3. Difference in development approach**

- The approach of developing a software vary from organization to organization.
- Different implementation is possible for same requirement set and It is depending on capabilities of developer.
- The best suitable approach is selected after observing circumstances.

➜ **4. Impossible exhaustive testing**

- Inspection and testing activity is aimed at finding flaws from products and development processes.
- Testing of all possible areas in product is practically impossible. High cost and time will be required to test all permutations and combinations.

➜ **5. Unaware about bad quality**

- As said earlier, testing of all algorithms, conditions etc. is impractical.
- Such areas remain untested and software is delivered to customer, which then used for longer time.
- Any problem in such areas will get discovered when particular situation occur.

➜ **6. Quality definition**

- Product quality depends on processes used for its development rather than rigorous inspection/testing activities applied on it.
- Finding and fixing defect will not improve quality of product and also will not guarantee defect free delivery.
- Good processes and procedures can only make a good quality software.

➜ **7. Quality objectives**

- There is a variation observed in quality objectives of different products.
- It depends on type of product, customer using it, time and circumstances. Quality objectives represents user expectations and their satisfaction level.
- They are defined on the basis of factors of quality i.e. 'must be', 'should be' and 'could be' for the applications.

Quality Factors

– Following are some of the quality factors.

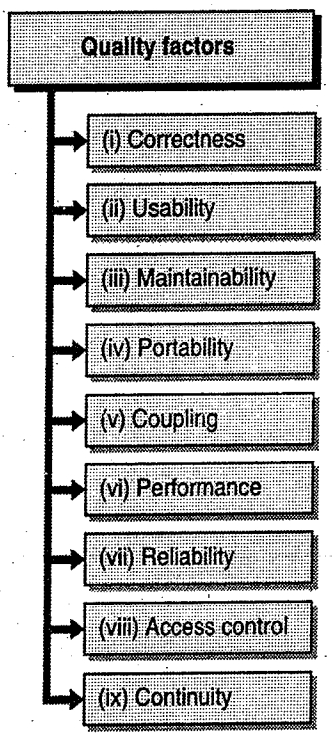


Fig. 1.27.2 : Quality Factors

➜ **(i) Correctness**

Defines the accuracy of outcomes

➜ **(ii) Usability**

Efforts needed to learn, operate, prepare input, and understand output.

➜ **(iii) Maintainability**

Efforts required to find and fix the defects.

➜ **(iv) Portability**

Efforts required to transfer software from one hardware/software configuration to another.

➜ **(v) Coupling**

Efforts required to integrate system components.

➜ **(vi) Performance**

Amount of resources needed to perform required functionalities.

➜ **(vii) Reliability**

System remain functional over the longer time period.

➜ **(viii) Access control**

Protection of system resources from misuse, destruction, modification etc.

➜ **(ix) Continuity**

Availability of essential procedures, methods, and backup information for recovering data, system and operations.

Syllabus Topic : Software Quality Management

1.28 Software Quality Management

– Quality management consists of set of planned and systematic activities (for development and maintenance) used to manage quality of product and services.

– It involves management of all input to the system processes in order to generate output matching to quality criteria.

– Quality management activities ensures that the software products and processes matches to defined standards, customer requirements etc. It presents three levels of handling problems which are as follows.

1. Correction

– It is considered as quality control approach. Defects are found in the system and are quickly fixed.

– Defect finding and fixing responsibility is handled by line function.

2. Corrective actions

– It is considered as quality assurance approach. Process related problems are detected and resolved by initiating corrective actions.

– Root cause analysis of defects is done and their introduction in the system is identified.

– Accordingly actions are initiated to stop occurrence of similar defects in future. Responsibility of corrective actions is handled by project leads.

3. Preventive actions

- After studying root causes of defects, other potentially weaker areas are identified.
- These areas can suffer from defect occurrence thus preventive actions are applied to them. Actions are initiated after checking similar scenarios or past history.
- Responsibility of preventive actions is handled by project manager (senior management).

Syllabus Topic : Why Software Has Defects ?

1.29 Why Software Has Defects?

- The software delivered to customer is not completely defect free in spite of taking all possible precautions, doing verification and validation activities.
- Following are some of the factors that can cause defect in the software :

1. Communication gap can make conversion of requirements into product very difficult. Majority defects are introduced due to wrong interpretation and understanding of requirements.
2. Requirements are changing dynamically and tracing those becomes difficult.
3. Customers are unaware about stated requirements and product development. Prototyping is used to give clear idea about it.
4. Software developers are very confident about their skills and abilities and do not think that they can commit any mistake.
5. Peer reviews and self reviews are not detecting any defects.
6. Technology (or application) is another source of introducing many defects in the system.

Syllabus Topic : Processes Related to Software Quality

1.30 Processes Related to Software Quality

Organization culture forms a strong foundation for Quality. There are different tiers of quality management and are discussed in following section.

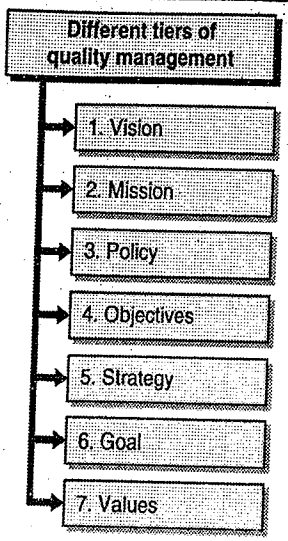


Fig. 1.30.1 : Different tiers of quality management

➜ **1. Vision**

- Every organization has its vision statement states ultimate aim it wishes to achieve in some time horizon.
- Vision statement is a brief idea about what organization wish to achieve and is established by management.

➜ **2. Mission**

- Organizations defining several initiatives are termed as mission statements.
- Success of these will help to achieve vision of organization. Mission statements having different lifespans.

➜ **3. Policy**

- Policy statement defines organization's way of doing business.
- It is generally defined by senior management and helps different stakeholders (customer, supplier, employee) to understand intent of organization.
- Multiple policies are possible in an organization which will help to achieve mission statements.

➜ **4. Objectives**

- Mission success and failure can be measured by using quantitative means termed as objectives of organization.
- Every mission statement must have at least one objective. It is defined in quantifiable terms along with duration for achieving it.

➜ **5. Strategy**

- Way of achieving the mission of organization is termed as strategy.

- Policies are converted into actions with the help of strategies. Strategies are defined using time duration, set of objectives and goals.

➜ **6. Goal**

- Small milestones are termed as goals used to achieve ultimate mission.
- Objectives are evaluated by reviewing milestones to understand whether progress is in proper direction or not.

➜ **7. Values**

- The way with which organization management think, behave, and believe is defined by values.
- Organizations are setting business principles based on values.

Syllabus Topic : Quality Management System's Structure

1.31 Quality Management System's Structure

Every organization has its own structure for **Quality management system** (QMS) based on needs and requirements. Following tiers forms typical structure of QMS.

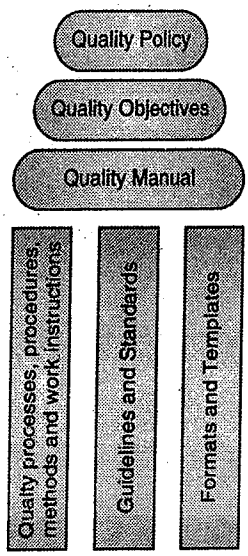


Fig. 1.31.1 : Typical structure of QMS for an Organization

1. Tier 1 - Quality policy

- It acts as basic framework in QMS that talks about intent, wish, and directions of management to carry out process activities.
- Management is the only driving force in an organization so its intent become most important.

2. Tier 2 - Quality objectives

- It helps to measure progress and achievements in a numerical terms.
- Quality improvements can be observed by looking at numerical values of achieved quality factors.
- These achieved values can be compared with planned one to find out deviation and to initiate further required actions.

3. Tier 3 - Quality manual

- It is also termed as policy manual which is established and published by management.
- It forms a strong foundation for quality planning at organizational level. It provides a framework for defining various processes.

Syllabus Topic : Pillars of Quality Management System

1.32 Pillars of Quality Management System

(1) Pillar 01

- It consists of quality processes, procedures, methods and work instructions.
- It is defined at organization level and project/function level by experts from respective functional area separately.
- Processes from organization level acts as umbrella under which project/function level processes are defined.
- Organization level processes may differ for different projects/functions.
- At project level, it is also defined as quality planning. Quality procedure must be synced at an organization level as per the quality manual.

(2) Pillar 02

- It consists of guidelines and formats used to achieve quality goals for products/services. It is used by project teams.
- Guidelines suggest a way of doing the things and can be overruled. Standards are defined by field experts and are mandatory ways of doing things.

- Customer defined guidelines can be treated as standard for a project. These two things needs constant revision to maintain suitability over time period.

(3) Pillar 03

- It consists of formats and templates used for tracking projects, function, department information within organization.
- It is used to maintain common understanding and consistency across all projects within organization.
- Templates can act as standards as they can be made mandatory while formats can act as guidelines as they can be suggestive.

Syllabus Topic : Important Aspects of Quality Management

1.33 Important Aspects of Quality Management

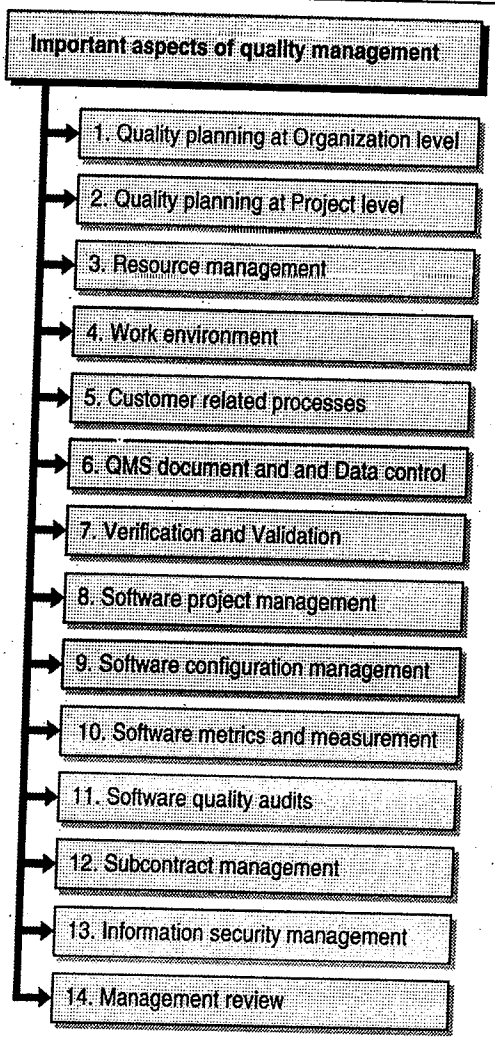


Fig. 1.33.1 : Important aspects of quality management

➜ **1. Quality planning at Organization level**

- Organizations creates quality plan that talks about vision and mission, policies, strategies, objectives, goals etc.
- It sets a framework for defining and implementing various practices, processes, recruitment, infrastructure, hardware, software etc.
- Achieved quality is checked against expected one mentioned in quality plan and if needed actions are employed.

➜ **2. Quality planning at Project level**

- Project quality plan must state project level objectives that are in sync with organization level objectives.
- It also defines various roles, responsibilities and actions.

➜ **3. Resource management**

- Basic resources can be people, methods, machines, materials etc.
- If the best combination of technology and process is given to people to work on it then planned results can be achieved.

➜ **4. Work environment**

- An essential input for a better product is its work environment.
- Bad environment can cause problem in achieving organization vision, mission, objectives.
- Work environment has two components i.e. internal (within organization) and external (outside the organization) environment.
- Good work environment builds strength of organization and achieves customer satisfaction.

➜ **5. Customer related processes**

- Capability of these processes is analysed with respect to services to customer and their satisfaction level.
- Processes can be from requirement analysis, design, delivery etc.
- Capable processes can only achieve good result. In case of deviating results some preventive and corrective action is initiated.

6. QMS document and Data control

- QMS are defined with the help of quality standards and quality models.
- Many organizations having specific customer requirements which can be an important input for defining QMS.
- Process improvement is done with the help of statistical process control and data management techniques.

7. Verification and Validation

- Verification and Validation activities are performed at every level of product development.
- Verification consists of project review, technical review, code review, management review etc.
- Validation consists of testing activities like unit testing, system testing etc.

8. Software project management

- Project management consist of tasks like planning, organising, staffing, directing, coordinating, controlling along with guiding, coaching and mentoring.

9. Software configuration management

- The product is tested (using verification and validation activities) on regular basis and continuously undergoes updation, integration.
- Configuration management takes care of creating products, their maintenance, review, and necessary updates.

10. Software metrics and measurement

- Project management defines various metrics program to measure process capabilities and product quality attributes.
- These programs takes current process data as an input to check its capability in delivering desired output.
- The deviations are corrected by applying necessary actions.

11. Software quality audits

- Software quality audit is conducted to analyze processes and products for their correctness.
- Based on result of audit required preventive and corrective actions are applied.
- Audits are conducted at predefined level and is of three types i.e. internal audit, customer audit, third-party audit.

12. Subcontract management

- A supplier is considered as an important stakeholder of organization. Proper methodologies are defined to build and maintain long-term relationship with suppliers.
- Input from suppliers are validated for their usefullness an effectiveness.
- Statistical control is applied supplier processes to avoid lose, delay, and service problems.

13. Information security management

- Three dimensions of information security are confidentiality, integrity, and availability.
- Information stored with applications and databases must be protected because this information acts as an input for continual improvement programs.

14. Management review

- It is conducted to analyze progress of various projects, departments, functions etc. that will be useful in achieving organization's vision, mission and objectives.
- Management must plan for future development in business. Management review must be a planned and systematic.
- Its input, methodology of conduction, and output must be clearly defined.

Review Questions

Q. Explain customer's view and supplier's view with respect to quality. **(Section 1.5)**

Q. Describe quality according Dr. Deming, Dr. Juran and Dr. Crosby. **(Section 1.2)**

Q. Describe cultural change requirement for quality improvement. **(Section 1.11)**

Q. What are the TQM principles of continual improvement? **(Section 1.8)**

Q. What are the core components of quality? **(Section 1.4)**

Q. Explain cost of quality in brief. **(Section 1.6.2)**

Q. Write a note on continual improvement cycle. **(Section 1.12)**

Q. Describe problem solving using software tools. **(Section 1.16)**

Q. How quality and productivity are related to each other? **(Section 1.20)**

Q. Explain the concept of 'q' and 'Q' organizations. **(Section 1.22)**

Q. Explain in brief various development models. **(Section 1.24)**

Q. Discuss product classification based on their criticality. **(Section 1.26)**

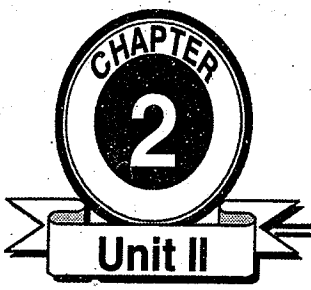
Q. Explain different types of products. **(Section 1.25)**

Q. Explain QMS structure for organizations. **(Section 1.31)**

Q. What are the major pillars of QMS? **(Section 1.32)**

Q. List and explain various reasons for software defects. **(Section 1.29)**

□□□



Test Planning and Management

Syllabus Topics

Review of Fundamentals of Software Testing, Testing during development life cycle, Requirement Traceability matrix, essentials, Workbench, Important Features of Testing Process, Misconceptions, Principles, salient and policy of Software testing, Test Strategy, Test Planning, Testing Process and number of defects found, Test team efficiency, Mutation testing, challenges, test team approach, Process problem faced, Cost aspect, establishing testing policy, methods, structured approach, categories of defect, Defect/ error/ mistake in software, Developing Test Strategy and Plan, Testing process, Attitude towards testing, approaches, challenges, Raising management awareness for testing, skills required by tester.

Syllabus Topic : Review of Fundamentals of Software Testing

2.1 Review of Fundamentals of Software Testing

- Software testing is used to find the differences between actual and expected behavior of system.
- It is comprised of different types of testing such as, it starts with feasibility testing, then contract testing, requirement testing, design testing, coding testing, and finally acceptance testing.
- Software development activities such as verification and validation are conducted at each stage of development. During verification, product is compared with standards and guidelines.
- Validation checks the final outcome with expected one. Conducting verification and validation activities comes under appraisal cost.
- When these activities are repeated through retesting and regression testing it comes under failure cost.

2.1.1 Historical Perspectives

- During initial days of software development no independent testing was considered.
- Developers were believed for perfect product development and customer was expected to use it.
- If any problem encountered then product was given back to development team for corrections.
- Development team corrects the product and again hand over it to customer.
- Later, Glenford Myer introduced the testing as a separate, independent phase in SDLC.
- According to Myer, testers were expected to use all combinations for testing the product. Primary intention was to make a product that will not fail during production.
- Testers were expected to break the system and will be eventually corrected in order to avoid its failure during use.
- This mechanism clearly separates debugging and testing and that has created independent testing community.

2.1.1(A) Debugging-Oriented Testing

- The developers were expected to debug the product while building it.
- Tests were not documented and heuristically conducted. Testing was considered to check correct working of implementation (positive testing).

2.1.1(B) Demonstration-Oriented Testing

- In this type software testers were introduced independent of development phase.
- Main aim was to show to customer that system is working. Approach was of positive testing and more advanced than debugging.
- Test cases were formed from SRS document. This approach demonstrated that the software performs as per the expectations of customer.

2.1.1(C) Destruction-Oriented Testing

- This approach was the basis for Myer definition. Tester were not supposed to check behavior of software under normal conditions.
- Instead they need to check that software does not fail under abnormal conditions (negative testing). Testers were on a mission of finding every minor flaw in product.

2.1.1(D) Evaluation-Oriented Testing

- This approach is followed at many places now a days. Product and process of software development is evaluated.
- In testing process, software is evaluated against quality factors or test factors derived from customer requirements. Producing defect free software is impossible but customer may accept product with some level of defect.
- This is known as level of confidence given to customer that product will behave as expected by customer.

2.1.1(E) Prevention-Oriented Testing

- The approach is followed in highly matured companies only. The testing process is considered as an opportunity for improving processes.
- Defects found are analyzed to know about process flaws and efforts are made to improve process in all aspects.
- This will improve the quality in software and reduce the dependency on testing phase.
- The cost will get reduced by creating quality product in first time only.

2.1.2 Testing Definition

- Software testing is used to detect defect by executing work product. It shows hidden defects so that it can be fixed and customer will not face any problem during use.
- It is not possible to produce defect free software. But if testing does not find any defect it means test scenarios and test cases design are incapable.
- Tester have to use positive testing and negative testing approach while performing testing activities. Negative testing is based on certain assumptions and risk that are mentioned in test plan while deciding test strategy.

☞ **Why testing is necessary ?**

- Interpretation and understanding of requirements is different for different people. With testing one can have another view about product to be developed.
- Developers are in a thinking state that whatever is developed is correct and as per requirements. But it is necessary and good practice to execute software in real life scenario to check it's working.
- Testing carried out against requirements will be beneficial to trace down and reduce the gaps between requirements, design, coding etc.
- A situation when independently tested units are integrated together but it is not working properly.
- Testers will try to break the system in order to check its proper functioning under normal and abnormal input conditions.

2.1.3 Testing Approaches

- The approach to testing is totally depend on several factors like, management, product to be developed, customer type, customer requirements, type of project, maturity of developing organization etc.

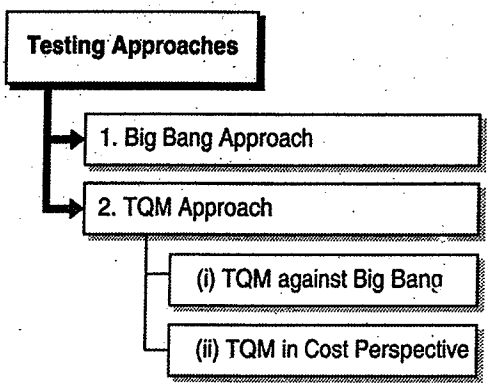


Fig. 2.1.1 : Testing approaches

- These testing approaches are mentioned under test strategy and are discussed in following section.

2.1.3(A) Big Bang Approach

- The testing is applied at the end of SDLC before product is given to customer for acceptance testing.
- It is also known as system testing and is the last phase as per waterfall model.
- In big bang approach, main focus is on testing black box functionalities against SRS document.
- It ensures the meeting of requirement and design specifications.
- This can show phase-wise defect distribution i.e defects in requirement, design, coding etc. phases.

Table 2.1.1 : Phase wise defect distribution

Development Phases	Defects in %
Requirements	58
Design	35
Coding	5
Others	2

- The big bang approach tests product at the end and may not be able to detect all defects from system.
- Applying all permutations and combinations of testing at the end of life-cycle is quite difficult due to schedule pressure. This may have cascading or camouflage effect.
- Even though defects are detected but it will not be that much effective.

2.1.3(B) TQM Approach

- The optimized and capable testing processes will not (less) produce defects and will not (few) leave any undetected defect in software when it is delivered to customer.
- Cost of defect remover after coding becomes 10 times of cost before coding.
- This cost comprise fixing of defects from requirement, design, coding etc.
- If defect is detected during acceptance testing then fixing such will be very much costly. Generally it is 100 times of cost before coding.

2.1.3(B)(i) TQM against Big Bang

- The organizations having good, optimized and capable processes are always produce consistent result.
- This will enhance productivity and effectiveness in development, and reduces cost of production significantly.

☞ **Stage 0**

- At this stage, organizations are quite mature and require no verification/validation activity to speak about quality.
- Cost invested in process definition, optimization, and deployment makes quality free.

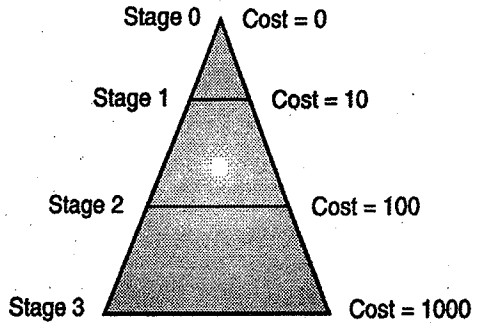


Fig. 2.1.2 : TQM cost triangle

☞ **Stage 1**

- In-spite of having quality environment, if in case defects are produced during development then organization may have verification process that will detect the defect in early stage before it gets spread in subsequent phases.

- A small cost is associated with this due to verification and fixing activity but it will save the subsequent stages from getting defect spread.
- This is called as appraisal cost and denoted as "10". This cost reduces profit of organization because of some rework, re-verification.

☞ **Stage 2**

- If some defects got escaped from verification process then a capable validation process can detect and fix those defects before the product delivery to its customer.
- Cost of validation is higher than verification. Once defect is detected, the stage from where it is originated can be identified.
- Hence it can be corrected from originating stage to detection stage.
- The cost is denoted as "100". Defects are detected and fixed before product delivery thus it won't affect customer feelings, faith and goodwill.

☞ **Stage 3**

- At the end, cost denoted as "1000" which is highest one. The defects are detected in acceptance test by customer.
- Fixing costs for such case is very high. There may be customer complaints, on-site fixing, selling with concession etc. which will affect customer relationship.

2.1.3(B)(ii) TQM in Cost Perspective

Main aim of TQM is to reduce cost of development and cost of quality through continual improvement. The term "quality is free" means quality must repay more than what is invested in it. TQM defines cost of development and quality using following three parts.

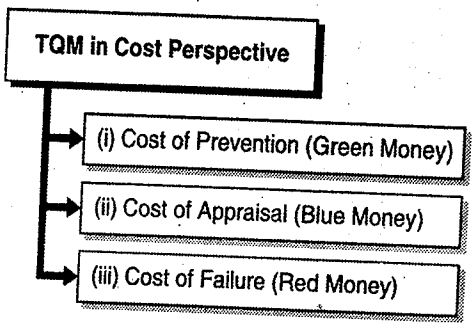


Fig. 2.1.3 : TQM in cost perspective

➜ **(i) Cost of Prevention (Green Money)**

- It is the cost associated with definition of processes, standards, guidelines, developing foundations for quality, training etc.
- The cost is incurred in performing quality work. The return on investment would be less inspection and testing, customer satisfaction, repeat orders. This will improve organization profit significantly.

➜ **(ii) Cost of Appraisal (Blue Money)**

- It is associated with cost incurred in development activities such as first time reviews, verification, validation and testing.
- Returns on investment is not possible but it helps to identify process capabilities and problems.
- First time testing helps to identify correctness in product so that it can undergo next stages.
- Initial cost of appraisal is increasing but as process gets more and more matured it gets lowered.

➜ **(iii) Cost of Failure (Red Money)**

- The cost is associated with rework, retesting, scrap, regression testing, sorting etc.
- It also represents waste produced during life-cycle that may result in customer dissatisfaction, loss of faith, business loss etc.

2.1.3(C) Characteristics of Big Bang Approach

- Big bang involves testing at the end of life-cycle that requires huge cost and efforts.
- It does not carry out any verification and validation activity in any prior phase.
- It also contains huge rework, retesting, scrap, sorting of software units and programs after building complete software.
- Special teams are deployed to find the defects and works only on correcting it in final testing.
- The testing process remain immature as no corrective and preventive actions are initiated to improve process capability.
- Regression testing highlights many issues such as correction of one defect in a component introduces many defects in its

dependent components. These are called as regression defects.

- Since delivery schedule is fixed, hence all requirement and design testing may not get covered during testing.
- Testing is done in a haste and multiple iterations of defect fixing are completed in shortest possible time.
- Due to this some defects flow to customer called as known defects. Some defects may not get declared at all.
- Due to tight delivery deadlines testing using adhoc, monkey, random, exploratory types will get carried out.
- Random testing does not give full coverage guarantee. Product is tested with intuition of tester and adhoc method.
- Generally positive testing is used to check correct functioning of product (known as second level maturity).
- Organizations following big bang approach are less matured and need to pay huge cost of failure multiple retesting, regression testing, and defect fixing cycles.

2.1.3(D) Observations in Testing

- Successful testing depends on skilled team, good development process, and type of customer.
- Applying verification activities during life-cycle can detect two-third of total defects.
- Cost of fixing is very less and stops stage contamination by defects.
- Applying validation through unit testing can detect three-fourth of remaining defects.
- The detected defects are fixed at that point only and stopped its re occurrence in system testing.
- Applying validation in system testing can detect 10% of total number of defects. System testing validates system level and design level requirements.
- Once exit criteria of system testing met, product is released to customer. Either system testing or acceptance testing is treated as certification testing.
- If testing is not managed properly, about 5% to 10% defects goes to customer.
- But with big bang approach, testing is able to detect only 5% of total defects.
- This means to achieve effective testing, it may need 18 system testing cycles that will incur huge cost.

2.1.4 Popular Testing Definitions

Software testing is not an activity performed at last stage of SDLC, instead it is a life-cycle operation.

2.1.4(A) Traditional Definitions

- Testing build confidence that software does what it is supposed to do (positive testing).
- Testing is an activity to evaluate attributes/capability of a product/process against user requirements.
- Testing is a process that demonstrates, product is error free. (acceptance testing).
- Testing is process conducted to confirm that product under testing is performing intended functionality as per SRS.

2.1.4(B) What is Testing ?

Software testing is driven by requirement specifications and design specifications. It is supported by test strategy and test approach based on assumptions, risks, and usage.

Following is included in testing process :

- An activity that identifies difference between expected and actual results by executing the product. This indicates presence of defect in process or in product.
- The process of detecting defects which are then fixed by developers by applying corrective measures. Root cause analysis of defects will help in improving processes.
- Testing ensures deviation from specification, misinterpretation of requirements and mismatch in requirements.
- The depth and width attained by testing ensures proper working of software under normal conditions.
- Testing evaluates any attribute of software, e.g. acceptance testing defines whether to accept the product or not.
- The coverage of testing gives measure of software quality (confidence about proper working).

- Testing activity evaluates processes used and provides an opportunity to make necessary improvements in it.
- Testing process is an attempt to verify that product under testing is satisfying defined requirements or not. Requirements can be understood with the help of customer.
- Testing confirms proper working of intended functionality of a software.

2.1.4(C) Manager's View of Testing

- Software product must meet customer's requirements (implied and defined).
- Software must be safe and reliable to use and must function under normal and abnormal conditions.
- Development and testing processes must be capable enough to detect defects and to impart confidence to customer.

2.1.4(D) Tester's View of Testing

- Testing must detect defects from development and testing processes.
- This will provide an opportunity to make necessary improvements in it.
- Testing is an attempt made to detect every defect in a work product which will be corrected eventually.

2.1.4(E) Customer's View of Testing

- Customer must be given with a product having no defects or less defects.
- Testing must remove all possible defects from work product. Testing must give a confidence to customer that software is protected from unwanted failures.
- Mean time between failures must be large enough. Testing must confirm that any legal or regulatory requirement is satisfied or not.

2.1.4(F) Testing Objectives

- Get the scenario where product is not doing what is supposed to do (referred as deviation from SRS).
- Get the scenario where product is doing things what it is not supposed to do. (referred as unwanted side effects or risks).

2.1.4(G) Basic Testing Principles

- Define expected results and actual results (after applying test cases) to find match between them. If not defect is considered, either in product or in test case or in test plan.
- Developers should not perform self-testing of their own programs. This is because no defects will be found in such type of testing.
- Carefully and completely observe test results as it will help in root cause analysis and may highlight weaker areas. Using this processes can be built in proper way with improved capability.
- Test the program using positive testing and negative testing approach.
- Design reusable test cases as they may be required during regression testing.
- Decide the targeted number defects while performing testing. Test should not be planned by assuming that product is defect free.
- The probability of getting more number of defects in a module is clearly indicating that many defects are already found in same module.

2.1.4(H) Successful Testers

- A successful tester is one who finds maximum number defects in a software product before its release.
- The tester must understand the problem areas, acquire all possible details, select the test data carefully.
- Tester must give confidence about test coverage and functionalities as per test plan. Tester must ensure that the risks associated with product are identified.
- The SWOT analysis of software and process must be conducted by tester.
- This will highlight weaker areas in process and avoid related defect from occurring again in future.

2.1.4(I) Successful Test Case

- Test cases must be capable of finding defects in the software before it is released to customer.
- Successful test case will reduce the probability of getting defect at customer end.
- If testing is not able to detect the defects then it's a loss to organization and customer as well.

Syllabus Topic : Testing During Development Life Cycle

2.2 Testing During Development Life Cycle

- This section is focusing on testing carried out on phases like : requirement, design, coding and testing.
- For understanding purpose, a waterfall model is considered as development methodology.

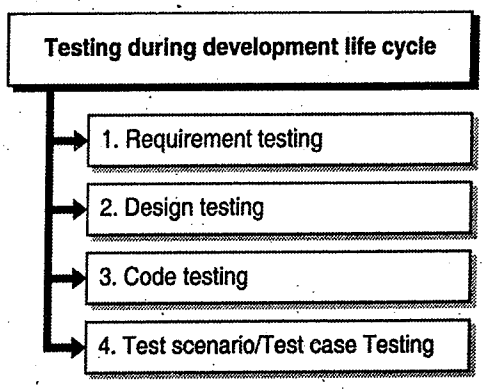


Fig. 2.2.1 : Testing during development life cycle

2.2.1 Requirement Testing

- Requirements given in requirement document must represent actors, transaction, information transfer from one system to another. It must be prioritized as - 'must', 'should' and 'could' by customer.
- It should give functional and non-functional testing scenarios. Requirement testing uses requirement document for having mock running of application to be developed. Use cases can be developed from requirements.
- Gaps in requirement can lead to defect. Business analyst must convert all implied requirements of customer into documented one.
- Requirement testing executes requirements to check consistency among them :

☞ **Characteristics (Requirements)**

- Requirement statement must be as per standard (e.g. performance, UI) and formats.
- Proper clarification about expected output by user (e.g. error messages, screen output etc.).
- Expected results must be measurable.
- Requirement traceability during entire SDLC is must.
- Testability of scenarios must be ensured.

2.2.2 Design Testing

- There are two types of system design :
  (i) Low level (detailed design) and
  (ii) High level (system architecture).
- In low level design, system and technical requirements are mapped with different entities involved to check adequacy of detail design.
- High level design testing includes mock execution of application along with its prerequisites.
- It is similar to develop flow-diagrams from design that tracks flow of information from beginning to end.
- The design is considered to be good if flow is complete otherwise it is an indication of design defects.
- Design verification is conducted by experts with the help of standards, templates, guidelines with a main objective of design review.

☞ **Characteristics**

- Requirement traceability
- Testability of design

2.2.3 Code Testing

- Developers writes code based on design guidelines, standard and design specification.
- Code testing also known as unit testing done using stubs and drivers. Code testing is done using code review.

☞ **Characteristics**

- Code should be readable, maintainable and reusable.
- Code should be testable in unit, integration, and system testing.
- Code should be traceable to requirements and design.

2.2.4 Test Scenario / Test Case Testing

- Test scenarios defines testing need of an application and are can be of type structural or functional.
- Scenarios are developed using requirements and design. Further they are converted into test cases.

☞ **Characteristics**

- Test scenarios should be clear and complete. It must represent processing and outcome in a clearer way.
- Test scenario must cover all defined requirements based on priorities.
- Test scenarios must be feasible so that can be created during testing.
- Test case document must cover all test scenarios.
- Test cases and scenarios must be prioritized.

Syllabus Topic : Requirement Traceability Matrix

2.3 Requirement Traceability Matrix

Requirements	High level design	Low level design	Code files / Stored procedures /TBLs	Test scenario	Test case	Test result

Fig. 2.3.1 : Typical structure of requirement traceability matrix

- Software application can be completely tracked and traced using Requirement Traceability Matrix (RTM).
- Various quality models and standards are using it. It is also known as blueprint of application that performs complete mapping of software application.
- It traces the application from requirement through design, code, test scenarios, test data, test cases, and test result.

☞ **Advantages**

1. Helps to understand software in a better way.
2. Answers question like-"what and how it is being developed?"
3. Traces non-implemented requirements and gap within requirements.
4. Identifies redundancy within application.
5. Test case failure can be tracked through requirement and design.
6. Requirement change that affects entire product can be traced.

☞ **Disadvantages**

1. Due to large number of requirements, creating RTM using some tools is a tough task.
2. Training people to use RTM is costly.
3. Managing various one-one, many-to-one, and many-to-many relationships requires large amount of efforts.
4. Change in requirement causes update in RTM, thus in design and code.
5. Development team may not understand usefulness of RTM.
6. Customer may not find RTM as a valuable unit and will not pay extra.

2.3.1 Horizontal

- It trace the application (in a forward direction) from requirement through design, code, test scenarios, test data, test cases, up-to test result.

- We can trace a requirement upon getting failure of a test case.
- Any design that is not having requirement will introduce a new feature and may be considered as defect.
- When any requirement is not traceable to design, then that requirement is not implemented in the application at all.
- Similar considerations are used during relationships of design-coding, coding-test scenarios, test scenarios-test cases etc.

2.3.2 Bidirectional

It traces application in forward and reverse direction also. CMMI model uses same approach.

2.3.3 Vertical

- It traces inter-dependencies in individual column of RTM. It focuses on parent-child relationship of requirements.
- One requirement may have several child requirements and vice versa. Similar type of inter-dependencies can be found in design, coding and testing.
- For example : calling function and called function shows inter-dependency in coding.

2.3.4 Risk Traceability

Risk	High level design / Control	Low level design / Control	Code files / Stored procedures / TBLs	Test scenario	Test case	Test result

Fig. 2.3.2 : Matrix for risk traceability

- With FMEA (Failure Mode Effect Analysis), developers can keep references to possible risks of failure in application.
- These risks can be traced down to requirements, design that offers greater control mechanism over risk and improves risk detention ability.
- This type of tracing helps user to know about accident prone areas in application and also where they are completely or partially protected from the failures.
- It will also help in identifying different controls that can be designed and used with application.

Syllabus Topic : Essentials of Testing

2.4 Essentials of Testing

- Software testing follows a disciplined approach that executes the software and identifies defects in it.
- The objective of software testing is to find all defects, eliminate it, and deliver a defect free software to a customer.
- Main principle of testing is that no software is a defect free but it is more acceptable when it has less risks.
- Testing team detects the defects which are then removed by development team before releasing product to customers.
- Testing phase must identify weaker areas or defect prone areas. Exhaustive testing is not possible wherein all permutations and combinations of a product will be tested.
- Testing can be viewed as SWOT analysis exercise where software can be developed using strength of development process and overcome the weaknesses in various processes.

☞ **Strength**

Areas like modules, screens, algorithms and processes like requirement, design, coding, testing etc. has very less number of defects and are considered as strong areas and processes.

☞ **Weakness**

- Areas where requirement are not mapped properly are considered as weaker one.

- These areas are at the boundary of failure and if something goes wrong it produces a defect.
- These areas must be analyzed to identify root causes of failures. Training and communication can be applied to such areas.

☞ **Opportunity**

- Some areas satisfies the customer requirements and further can be improved.
- This improvement represents organization's ability to help customer for doing something better.

☞ **Threats**

- Threats are the defects that will lead to software failure. Usually it occurs due to problematic processes and organization are trying harder to make them stronger.
- Such a problematic process can lead to customer dissatisfaction.

Syllabus Topic : Workbench

2.5 Workbench

- The term is derived from engineering production domain, where it receives an input from previous workbench and gives output to next workbench.
- The software development and testing has many interlinked activities.
- Output of previous activity is taken as input for current activity, and its output becomes input for next activity.
- Workbench consists of defined procedures to assess work and produced output.
- The work can be requirement analysis, design, coding, testing.
- Organization process database consists of methods, procedures, standards, guidelines etc. to be followed while working, to improve process to achieve customer satisfaction.
- If the standard set is observing any deviation in current result and expected one, then work product is said to be defective and process used are reworked for correctness.

☞ **Tester workbench**

- It consists of testing process, tools, guidelines, standards etc. for every workbench there is defined entry criteria, process for checking work and an exit criteria.
- Definition of each workbench may be written in test plan.
- Following are the eight possible tester workbench examples :

1. Creating test strategy	2. Creating test plan
3. Writing test scenario	4. Writing test cases
5. Test execution	6. Defect management
7. Retesting	8. Regression testing

- Typical workbench for system testing execution :

1. **Inputs to tester workbench :** Test scenarios, test cases, working products, documentation, test environment, test plan, note containing some issues.
2. **Do process :** A normal tester is guided during testing as per process defined organizational process database. Testing is carried out as per test cases and test procedures.
3. **Check process :** It checks correctness of "Do Process".
4. **Output :** It must be generated in requires test report and test log format.
5. **Standards and Tools :** Standards are defined as steps to be followed. Tools are related to configuration management, defect management, regression testing etc.
6. **Rework :** If "check process" observes failure in "do process" results then rework is initiated.
7. Do process and check process can be suspended if serious problems are observed during its functioning. It is known as "suspension criteria". Once problems are resolved both can be restarted. It is known as "resumption criteria".

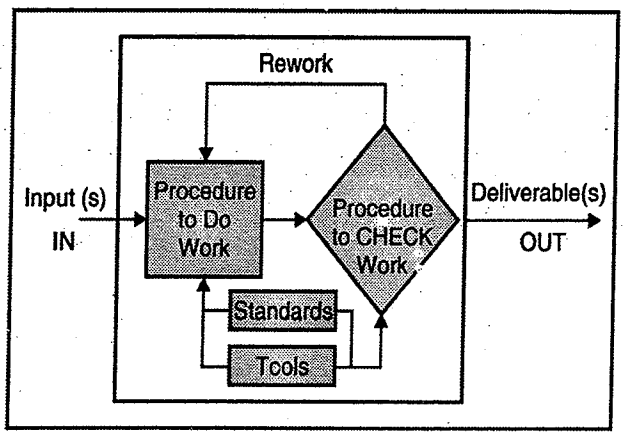


Fig. 2.5.1 : Typical workbench

Syllabus Topic : Important Features of Testing Process

2.6 Important Features of Testing Process

1. Testing is considered as constructive destruction of a work product. The intention behind destruction is finding defects and deliver a good quality product to customer. Application is tested with various input conditions. A successful tester is one who break the application in all aspects. During negative testing, tester has to think in a destructive way to build required test scenarios.
2. Generally a defect-free product does not exists. Tester need to perform thorough testing to identify defects and possible risks in product. The presence of defect highlights the fact that development process is problematic.
3. If tester is unable to find defect then it is considered as unsuccessful one. This will cause failure of software at customer end.
4. The root cause analysis of every defect is important, because it highlights the weaker areas in work product which need to be strengthen. This defect analysis would improve the development process and leads to higher customer satisfaction. Testing is considered as investment done by customer and organization.
5. Generally less time, money, resources and support is provided to testing activity across all organizations. The thinking behind this is with less testing there will be less defects, less rework, less scrap, less corrective actions etc. if a less tested product is delivered then there are high chances of its failure at customer end.
6. Test strategy defines the effectiveness and efficiency of testing. Test cases should be categorized according to installation testing, smoke testing, sanity testing. Test scenarios and test cases will reduce the probability of software failure in production environment. Organization may define test cases with priority ranging from high , low and medium .testing should not be treated as end process, instead it should be imparted in every phase of SDLC.
7. The root cause analysis of defect should be done as soon as it is detected in particular phase. This will limit the phase contamination means migration of defect from one phase to next subsequent phases. Fixing defects in later stages is very costly approach.
8. Testing is not defect finding process and fixing process. It must be considered as defect prevention process. Such defect prevention mechanisms must be installed and operational. The defect are analyzed thoroughly and are prevented from occurring again and again.

Syllabus Topic : Misconceptions

2.7 Misconceptions

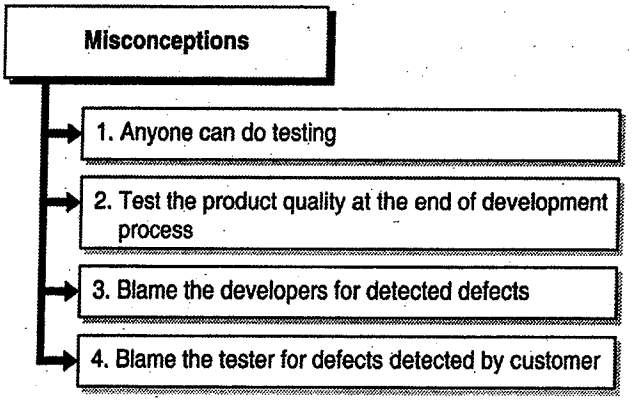


Fig. 2.7.1 : Misconceptions

➜ **1. Anyone can do testing**

- Many organizations put less experienced people (in testing domain) at the bench of testers.
- It is highly impossible to define effective test cases, test scenarios, test data with such inexperienced and less skilled tester.

- Organizations should not think that investing on training and skills of a tester is waste of time and money. This will lead to disaster at customer end once product is delivered.

➜ **2. Test the product quality at the end of development process**

- In such approach, system testing and acceptance testing are considered as qualification criteria of product delivery.
- Only few test cases are applied from huge set. This will lead to have hidden defects in software and such a product will not fulfill requirements and expectations of customer.

➜ **3. Blame the developers for detected defects**

- Developer should not be blamed always because the code is developed from design of a product using set standards and guidelines.
- Sometimes defects are present due to ineffective development processes.
- Management must ensure proper and good input should be provided to development team.

➜ **4. Blame the tester for defects detected by customer**

- Tester is executing few test cases and do not check intended functionality of software.
- Generally 100% testing is not possible. If no defect is found during testing does not mean that software is free from defects.
- Some defects are found during real time use of product.
- Testers must learn from experiences, must perform root cause analysis and avoid recursive occurrence of defects.

Syllabus Topic : Principles

2.8 Principles

1. Developers or programmers must avoid self review or self testing. Everybody has soft-corner and love towards work-product made by him/her. Though self review is good method but will not be effective in finding defects with high level. Second opinion will always play an important role in improving product quality.
2. Inspect test result thoroughly to know weaker areas in product and development process. Management and development team will understand areas to perform root cause analysis.
3. Corrective and preventive action must be initiated to improve product and development process. These actions must be applied recursively to have a better product.

Syllabus Topic : Salient Features of Software Testing

2.9 Salient Features

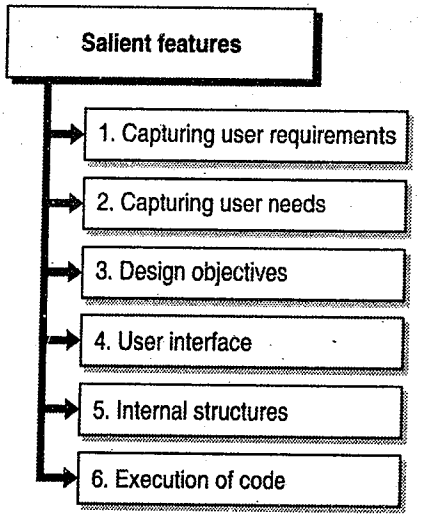


Fig. 2.9.1 : Salient features

➜ **1. Capturing user requirements**

- User requirements (functional, non-functional, system, operational, technical, legal etc) forms foundation of a software product.
- Using these requirements tester designs test scenarios and test cases.
- In general, user is able to state only functional and non-functional requirement but the other requirements must be defined by organization.

➜ **2. Capturing user needs**

- User needs may be different from SRS document. It may include implied requirements, process requirements, present and future requirements etc.
- Understanding and interpreting these requirements is the responsibility of organization.

➜ **3. Design objectives**

- Design of software decides an approach to be followed for building it. The approach is selected based on framework used for development and testing.
- It is necessary to document about-how functional, UI, performance and other type of requirements will be integrated in design of system and how it will be achieved in development and testing phase.

➜ **4. User interface**

- UI should be simple and easy to use/understand. User should understand what he is supposed to do and what system is doing.
- UI defines user's ability to interact with system, receive error message and take action accordingly etc.

➜ **5. Internal structures**

- Internal structure is based on system design, guidelines, standards etc.
- These guidelines and standards may be defined by organization or by customer.
- It may discuss about re-usability, commenting, nesting etc. Organization must understand benefits, costs, disadvantages of different approaches.

➜ **6. Execution of code**

- Work product is executed to ensure it is performing intended task as per defined in SRS document and interpreted from product design.
- Negative testing ensures that product is not doing what it is not supposed to do.

Syllabus Topic : Policy of Software Testing

2.10 Policy of Software Testing

- Test policy defined by senior management or by customer based on type of organization. In project organization, it is defined by the customer.
- In product organization, it is defined by senior management.
- The policy document covers all aspects of testing and a framework of testing, both aimed at achieving customer satisfaction.

Syllabus Topic : Test Strategy

2.11 Test Strategy

- It is also referred as "test approach". It includes different actions and ways to be followed to achieve test policy.
- In general there is single test policy at the organization level but test strategies may differ from customer to customer, product to product.
- Examples of test strategies are as follows :
  1. Coverage Definitions (requirement coverage, function coverage etc.)
  2. Levels of testing (from requirement to acceptance phase)
  3. Amount of Manual Testing & Automation Testing
  4. Ratio of Developers to Testers.

Syllabus Topic : Test Planning

2.12 Test Planning

- Test planning is the very first activity of testing team that is used throughout the SDLC.
- It is defined within the framework created by test strategy and established by test policy.
- Test plan contains testing at various stages during SDLC. It talks about assumptions, constraints, limitations, and risks associated with testing.

☞ **Characteristics**

1. Test planning should predict the number defects to be detected during testing. It it finds those many defects then it considered as a good test plan. It must define the required number of testing efforts (testing iteration).
2. No software is defect free, hence if the test plan is unable to find any defects then there is problem in designed test cases, test data etc. Such a test process is termed as defective. On the other hand, if defect found are more then development process is problematic. With every found defect, probability of its finding by customer is also reduced.

3. Testing process is said to be a successful when it identifies number of defects in the application under test. It is not intended to certify that the development process is 'good' or 'bad'. Successful testers can find all possible frequently occurring and severe defects and contribute to deliver a successful application.

4. Testing process should not be applied at the end of SDLC. Instead it should be the part of every single stage of SDLC. Application should be certified by customer not by developer or tester. Software testing involves two processes i.e. verification and validation.

Table 2.11.1 : Difference between verification and validation

Sr. No.	Verification	Validation
1.	Verification answers the question - "Are we developing this product by firmly following all design specifications ?" (Are we building the product right ?).	Validation answers the question - "Are we developing the product which attempts all that user needs from this software?" (Are we building the right product?).
2.	It is the process of evaluating products in the development phase to determine whether they are meeting the specified requirements.	Validation process is focusing on the final product. It evaluates software during the or at the end of the development process to check whether it satisfies specified business requirements or client's needs.
3.	In the process of verification, the product is referred as, requirement specification, design specification, code, user manual, or even the final product.	It also includes comparing test results to expected results.
4.	It comprises various activities such as, requirement reviews, design reviews, code reviews that check the correctness of a development phase.	It comprises various activities such as, Unit testing, Integration testing, System testing, User acceptance testing.
5.	More concerned with white box testing or static testing.	More concerned with black box testing or dynamic testing.
6.	Can find about 60% defects.	Can find about 30% defects.
7.	Approach is prevention. based.	Approach is detection. based.
8.	Includes coding guidelines, commenting, nesting, variable declaration etc.	Includes - performance, security, stress and volume testing.

Syllabus Topic : Testing Process and Number of Defects Found

2.13 Testing Process and Number of Defects Found

- The main objective of testing process is to find more number of defects. There will be very less probability of defects detected by customer after having effective testing of application.
- As more number of defects detected, there are high chances that still we can get more number of defects.
- This is based on principle saying that no application is defect free and every testing team has efficiency to detect defects.
- After adequate amount of testing, the number of defects found will always indicate possibility of number of defects exist in an application.

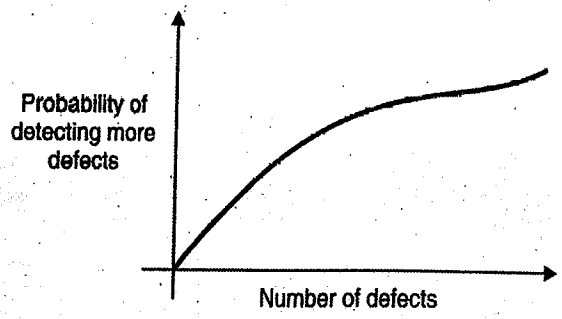


Fig. 2.13.1 : Defect trend in an application

Syllabus Topic : Test Team Efficiency

2.14 Test Team Efficiency

- It is concerned with ability of test team of finding defects in application under test. It is an important consideration from management and development team point of view.

- Test managers and project managers are always keeping eye on their team's testing efficiency. Less efficiency requires more number of iterations of testing.
- Suppose an application has 200 defects and testing team is able to detect 180 defects from it, then the team efficiency is considered as 90%. Ideally no team has 100% efficiency, but it must be close to it.
- Team efficiency deviating from this mark will not be considered as a good test team and becomes highly unreliable.
- Test team efficiency is depend on organization culture and is improved by having considerable efforts from organization.
- For example, suppose X is the number of defects deliberately introduced by development team. Y is the defects detected by test team.
- Z is the number of defects detected by test team but not belonging to set of defects from X. Test team efficiency can be given by ratio : (Y – Z) / X.

Syllabus Topic : Mutation Testing

2.15 Mutation Testing

- It is used to determine efficiency of test program and test case in detecting defects.
- A program module is developed, test cases are designed and executed on this module that will find some defects.
- Now original program is changed and some defects are deliberately added into it. Same set of test cases are applied on this changed program.
- This process is called as mutation whereas changed program is called as mutant.
- For example, suppose X is the number of defects deliberately introduced by development team. Y is the number defects detected by test cases in original module.
- Z is number of defects detected by test cases in mutant. Test case efficiency is given by ratio : (Z – Y) / X.

Ideally, the test case efficiency must be 100% but it is not possible due to following reasons :

1. **Camouflage effect**
   - This means that, two defects are compensating each other. Test cases may not be able to detect hidden defects.
   - Similarly tester unable to find defects introduced by developer.
2. **Cascading effect**
   - Some defects may get introduced into system due to existing one.
   - Such defects may be seen by tester as these are not introduced by developer.
3. **Coverage effect**
   - Some defects arise in a portion which is not considered for testing (coverage testing).
   - No test case is written for that area leads to generate an error. Testers are unable to detect such defects.
4. **Redundant effect**
   - Some parts are not getting executed due to certain conditions.
   - Defects belonging to these parts will not be detected by tester at all as such parts are not getting executed.

Syllabus Topic : Challenges in Testing

2.16 Challenges in Testing

1. Unclear, incomplete, inconsistent, non-measurable requirements leads to problem in designing test scenarios and test cases.
2. Problematic configuration management system where change in requirement is not communicated properly to test team.
3. Wrong documentation and interpretation of requirements by business analysts.
4. Testers unable to understand program logic due to lack of technical knowledge.
5. Error handling is difficult as it requires error messages and control mechanism (detective, corrective, suggestive, preventive etc.)

6. Badly written, unreadable, non-maintainable codes/programs leads to many defects.
7. Bad architecture creates complex application code and adds many defects.
8. If more defects are found, more testing effort are required, this delays the application delivery and testers are blamed for this.
9. If application is not tested properly and delivered within time that receives constant customer complaints, again testers are held responsible for this.
10. Management may have problem in understanding testing approach that may create thinking as testing is an obstacle before application delivery.

Syllabus Topic : Test Team Approach

2.17 Test Team Approach

- Test team formation is depend on type of organization and type of product to be developed.
- If application demands or if management does not find importance of testing team decides the existence of it in an organization. It consists four different approaches and are as follows :

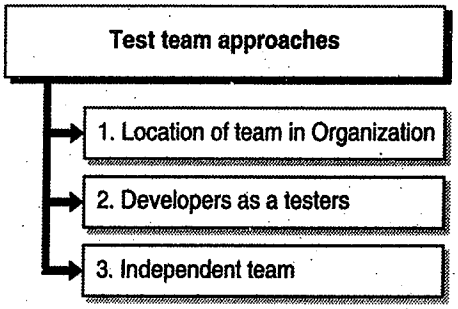


Fig. 2.17.1 : Test team approaches

1. Location of team in organization

- Test teams locations are defined in test policy. It depends on organization, project and customer to decide location of test team.

Following are the three methods :

(A) Independent team

Test teams are independent from development activities and need not to report to development manager. Instead they report to senior management or to customer. They have test manager for leading them throughout the project.

Advantages

(i) No deliver pressure.

(ii) Have sufficient time to execute multiple iteration and conduct full coverage.

(iii) No pressure of "not finding" defect.

(iv) Can have another view about product apart from development team.

(v) Expert guidance and mentoring available from test manager.

Disadvantages

(i) Testers always try to break system as there is conflict between mentality of testers and developers.

(ii) Testers won't get proper understanding of development process.

(iii) Tester is treated as outsider.

(iv) Management may be somewhat inclined towards development team creates feeling of no value in organization.

(B) Team reporting to development manager

Test team is involved from start of project to its end. Teams are reporting to development manager.

Advantages

(i) Better cooperation can be found between development and test team.

(ii) Involvement in development, verification, validation activities.

(iii) Gets good understanding of requirements and development process.

Disadvantages

(i) No expert advice and mentoring is available.

(ii) Development manager is more inclined towards development team.

(iii) Defects identified may be treated as hurdles in project delivery and are neglected.

(iv) Testers approach of looking at a software may get changed causes in reduction of defect finding ability.

(C) Matrix organization

It combines advantages of both the fore mentioned methods. Test team reports to development manager regarding functionality, and will report test manager for administrative works.

➜ **2. Developers as a testers**

- Developers work in the early phases of SDLC and may act as tester in later phases.
- It is required when application is heavy in technological regard and has less structured testing in it.

☞ **Advantages**

(i) No extra knowledge of testing is needed.

(ii) They have better understanding of design and coding hence can test application easily.

(iii) Developers can write automation scripts also by acquiring needed skills.

(iv) Cost overhead reduced by not having dedicated test team.

(v) Developers are able to find defects.

☞ **Disadvantages**

(i) Developers will not be more concerned about performance testing.

(ii) Approach of finding defect will be missing.

(iii) They may concentrate more on development activities.

(iv) Testing requires destruction skills that may be lacking.

➜ **3. Independent team**

A dedicated test team is established having required knowledge of testing and ability to detect defects.

☞ **Advantages**

(i) More focus on test planning, test strategy, test deliverables etc.

(ii) Independent view about product can be put forth after reviewing requirements.

(iii) Skilled workforce is acting in testing phase.

☞ **Disadvantages**

(i) Separate team results in additional cost overhead.

(ii) Frequently conflicts may occur between development and test team.

(iii) Test team need to update and upgrade their knowledge.

➜ **4. Domain experts for testing**

System and acceptance types of testing requires domain specific testing. Domain expert may use their expert knowledge in such cases.

☞ **Advantages**

(i) Software is tested from user's perspective.

(ii) They may interpret user requirements in a correct manner and context.

(iii) They may help development team to understand defects and customer expectations.

☞ **Disadvantages**

(i) Fail to understand exact requirement of user because of lack of knowledge in diverse areas.

(ii) More costlier approach.

Syllabus Topic : Process Problem Faced

2.18 Process Problem Faced

- 'Q' types of organization considers that the defect in the product are because of incorrect and faulty processes.
- Defects introduced due to wrong development and testing processes.
- A faulty testing process would give problems like - 'defect not found', 'duplicate defect found' , 'defect out of scope', 'defect cannot be reproduced' etc.

Following section explains major four components of process and problems associated with them.

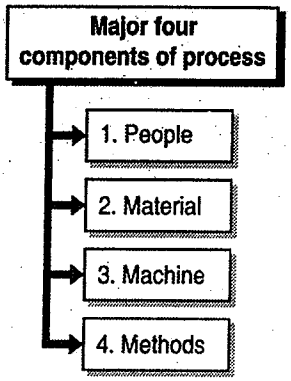


Fig. 2.18.1 : Major four components of process

➜ **1. People**

- Lack of proper skill-set, domain knowledge, development and testing process knowledge.
- Capabilities of test people (business analyst, test manager, tester, customer etc.) creates problem in development and testing.

➜ **2. Material**

- Documents like; SRS, development standard, test standard, guidelines that would add knowledge about system perspective are incomplete and unclear.
- Faulty test plans, project plans, organizational process documents.
- Non availability of test tools, defect management tools.

➜ **3. Machine**

Problems related to improper and wrong testing environment.

➜ **4. Methods**

Methods used for test planning, design of test scenario, writing test cases, selection of test data, risk analysis may not be a proper one.

☞ **Economy of testing**

- As testing phase progresses, more number of defects gets uncovered and probability of problems that customer may face will get reduced.
- This will increase the testing cost. Cost of customer dissatisfaction is inversely proportional to efforts required during testing.
- More testing efforts will reduce the customer dissatisfaction and cost of testing is always increase in an exponential way.
- If a graph is plotted for both the above conditions the two curves will intersect each other.
- This is shown in Fig. 2.18.2. Intersect point indicates optimum testing area. Before this point, it is "under testing" area where defective product reaches to customer and cost of dissatisfaction is higher than testing cost.
- Whereas, area after the intersection point is "over testing" where cost of customer dissatisfaction is less than testing cost.

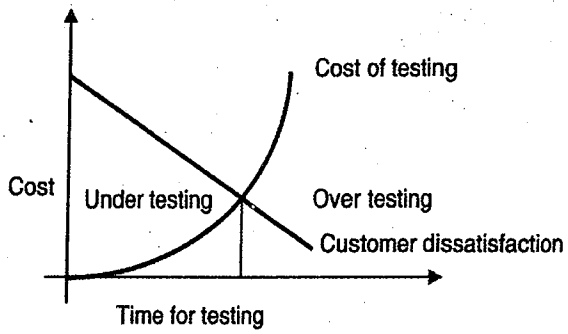


Fig. 2.18.2 : Cost of testing and cost of customer dissatisfaction

☞ **Guidelines for cost of testing**

- Test team efficiency - ability of test team to find defects.
- Defect fixing efficiency - ability of development team to fix defects.
- Defect introduction index of development team - defects introduced after fixing earlier discovered defects.

☞ **Guidelines for customer dissatisfaction**

- Highly depends on customer-supplier relationship.
- If organization has delivered successful projects in past the dissatisfaction curve remains parallel to Y-axis otherwise it will remain parallel to X-axis.
- It is also depends on type of application. For example, critical applications does not accept minor problem even.

Syllabus Topic : Cost Aspect

2.19 Cost Aspect

- Cost of quality comprises cost of testing also. Generally testing is considered as costlier phase.
- So organizations are always trying to minimize this cost. Efforts required in development and testing phases are converted into some rates that get added into total cost of product.
- For human resources, cost of efforts changes as per the role played by individual.

For example,

Project duration (in months) = 10

Number of working days per month = 22

Number of working hrs per day = 08

Number of working people = 100

Conversion rate (in Rs.) = 500/- per person hr.

Cost of development and testing is as follows -

Total efforts spent on project = $(10 \times 22 \times 08 \times 100)$

Total efforts spent on project = 1,76,000 hrs.

Total cost = $1,76,000 \times 500$

Total cost = Rs. 8,80,00,000

To arrive at sales price, few more costs are get added which are as follows :

Contingency = 10% ,

Overhead = 10%,

Expected profit = 20%

Sales Price = $8,80,00,000 \times 110\% \times 110\% \times 120\%$

Sales Price = Rs. 12,77,76,00

- The above scenario is not a standard one. Usually development project does not have same number of people in all phases (in above example, 100 people were working for all 10 months).
- Initially number of people is less, as phases are passed number of resources increased and once peak tasks are over it again goes down.
- This is common for development and testing phases. Fig. 2.19.1 represents requirements for a development project.
- For maintenance phase number of resources remain constant for long time.

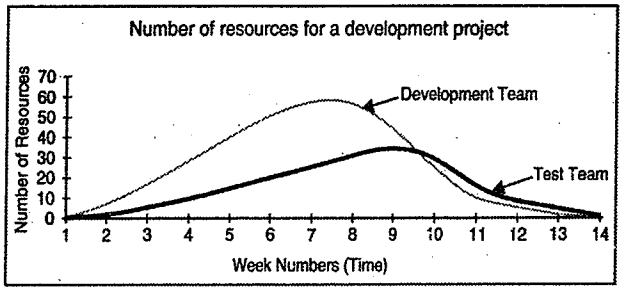


Fig. 2.19.1 : Resource requirement for a development project

Cost of development include following :

- Requirement collection, analysis, understanding etc.
- Cost of design (high level and low level).
- Cost of coding, integration, product delivery etc.

1. Assessment of cost of testing

- It is considered as function of process maturity and testing maturity.
- Cost of testing is above the overall cost of project development. It depends on following factors :

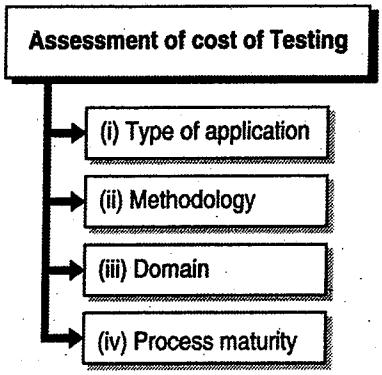


Fig. 2.19.2 : Assessment of cost of testing

➜ **(i) Type of application**

- Testing efforts increases in an exponential way with the increase in size of application (according to Rayleigh Curve).
- Organization must use historical data to evaluate their own curve. Cost of testing is defined as follows :

Cost of testing = K f(x),

where,

K = constant,

f(x) = function of application size,

x = size of application.

➜ **(ii) Methodology**

- Methodologies used for development and testing affects the effort and cost. e.g. Waterfall model requires less testing efforts, whereas agile and iterative model need huge testing efforts (due to regression testing and change in requirements respectively).
- Phase-wise testing or life-cycle testing gives less cost as compared to testing conducted in last phase.
- Use of automation testing may minimize the cost of testing.

➜ **(iii) Domain**

- Domain and criticality of application is very important in deciding cost of testing. More critical application requires perfect test results.
- Applications are having implications of law, regulation, money, rules etc.
- OOP has wide advantages over traditional programming. Unit testing is having high weight-age than integration testing.

➜ **(iv) Process maturity**

- Software application development has not achieved a level of maturity to produce defect-free products.
- Testing process must identify the defects in product. In first iteration, all defects are detected and fixed.
- In second iteration, confidence is increased that defects are fixed and product will not fail.

2. Cost of prevention in testing

Cost is invested in preventing defect from entering into the system. Following are the two ways.

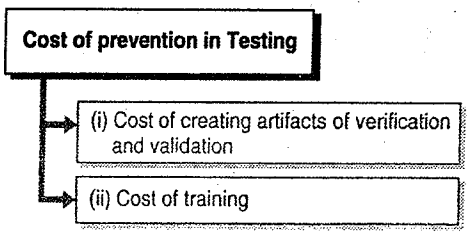


Fig. 2.19.3 : Cost of prevention in testing

➜ **(i) Cost of creating artifacts of verification and validation**

It includes cost associated with - creation of test plan and quality plan, review conduction, checklist preparation, formation of testing guidelines, writing test scenarios, test data and cases, writing scripts for automation etc.

➜ **(ii) Cost of training**

It includes cost for some of the following training related to - domain, project specific, organization level, tools, test process etc.

☞ **Cost of appraisal in testing**

It includes cost incurred for doing verification activities, validation activities, unit testing, integration testing, system testing, reviews, walkthrough, inspection etc.

☞ **Cost of failure in testing**

It includes cost of performing retesting, regression testing, re-reviews, activities above first time verification and validation etc.

Syllabus Topic : Establishing Testing Policy

2.20 Establishing Testing Policy

- An effective testing will be carried out with a properly planned testing efforts.
- This process is driven with the help of test policy, test strategy, test planning etc.
- Test policy defined by management that should talk about test objectives (the way testing is carried out), test deliverables, and finally customer satisfaction.

☞ **Test strategy includes**

- Steps required for effective testing.
- Required number of test cycles.
- Defect capturing and reporting.
- Test tools and environment.

Test objectives includes

- Definition of objectives in measurable terms.
- Objectives about requirement coverage, code coverage etc.
- Predicted number of defects.

Test plan includes

- List of decided test objectives.
- Methods used for defining test scenarios, test cases, and test data.
- Way of representing the test output.
- Way of doing retesting.

Syllabus Topic : Methods

2.21 Methods

- Management establishes the methods applied for testing efforts at organization level.
- These methods are generic in nature and can be customized as per test plan that suits to specific project needs. Generally it includes
  - Which parts to be tested or not tested ?
  - Tools to be used.
  - Defect tracking and reporting mechanism.
  - Communication methods.
  - Decision about automation.
- Management directives are generally defined in test strategy which may be discussed with customer/user to get their views about testing.
- It is done through meetings or MOUs (Memorandum of Understanding).
- Customer must be communicated about cost of defect detection, fixing of defects and acceptance criteria.
- Data and input methods provided by customer must be analyzed for correctness.

Syllabus Topic : Structured Approach

2.22 Structured Approach

- Testing should not be treated as a last stage activity in SDLC, instead it must be a part of entire life cycle.
- If it is applied in last stage then cost will be high and result will be inaccurate. Following four components of wastes are involved in this type of testing.

1. Wrong product and testing is done if wrong specification is used for development.
2. Sometimes correct specification may get interpreted in a wrong way in design and coding stages.
3. The white box testing and review processes are not detecting defects hence it will get detected at customer end, leads to customer dissatisfaction, rework, retesting.
4. Testing cost will be very high, if attempt is made to find all defects in a product.
5. Effective reviews can reduce cost in development and testing. If software quality is left to black box testing or system testing then it will result in high cost of testing and customer dissatisfaction.
6. Cost of fixing defects in last phase is very high as there are many phases between event of defect introduction and defect detection.
7. Corrections in specification, design, code, regression testing etc. is very costly approach. This will result in disturbing schedule, waste of efforts, and customer dissatisfaction.
8. Many times fixing a defect in one part may introduce defect in another part in a product.
9. Software must be tested again and again to ensure its correctness with respect to all part (changed and unchanged). Regression testing is an useful approach in such cases.

Syllabus Topic : Categories of Defect

2.23 Categories of Defect

- Defects are categorized based on various criterion. These categories are created based on type of organization, project or customer. These categories are defined in a test plan.

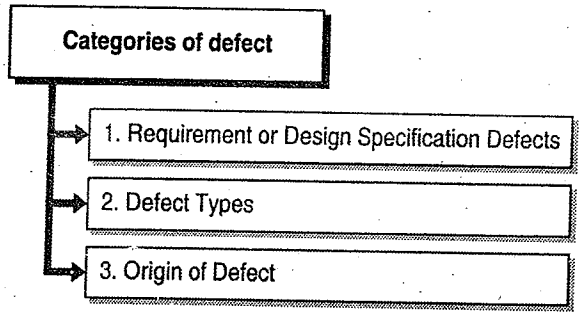


Fig. 2.23.1 : Categories of defect

➜ **1. Requirement or design specification defects**

- This type of defect represented by variation from product specification that is defined in requirement specification and design specification.
- Defects from this category are responsible for "Producer's Gap".
- Sometimes business analysts are unable to identify customer's expectations/needs properly which will represent a variation in the form of implied type of requirements. Defects from this category are responsible for "User's Gap".

➜ **2. Defect types**

- Wrongly implemented specifications as per the understanding of developer will vary from customer's expectations. It is termed as "misinterpretation of Specification"
- Specifications are missed in final product that are defined in specification document earlier.
- Extra features present in a product that are not supported by specification document.

➜ **3. Origin of defect**

- Wrong requirements given by customers. It is due to inability of customers to put or specify requirements in correct words.
- Wrong interpretation of stated requirements by Business analysts.
- Wrong architecture created by System design architect. This is due inability in understanding requirements, communication gap etc.
- Due to incapable development processes, incorrect specifications, guidelines and standards may get used.
- Lack of developer's skills in understanding the design may introduce errors in code.
- Due to improper product design, data entry error may be caused by users.
- Some defects are not detected during testing and may get identified at customer end.
- While applying corrective action to one part of product, defect may get introduced in another part of same product.

Syllabus Topic : Defect / Error / Mistake in Software

2.24 Defect, Error and Mistake in Software

1. Defect

- Issue identified in black box testing.
- Sometimes identified by customer or user.
- Takes longer time to fix this issues
- Most costly
- Problems and solutions are properly documented and used for improving processes.

2. Error

- Issue identified through unit testing.
- Generally identified in internal environment only.
- Needs reasonable amount of time to get fixed.
- Less costly than defect but more than mistake.
- Problems and solutions are properly documented and not used for improving processes.

3. Mistake

- Issue identified through document review, peer review etc.
- Less costly than above two types.
- Takes very less time to get fixed.
- Problems and solutions are not properly documented.

Syllabus Topic : Developing Test Strategy and Plan

2.25 Developing Test Strategy and Plan

2.25.1 Test Strategy

Test strategy defines how the test team will carry out testing task. Test plan contains the information of test strategy. It has two components and are as follows :

(i) Test factors needed in particular phase of development.

(ii) Test phase corresponds to development phase.

Following are the stages through which test strategy is developed.

1. It is necessary that test team must identify test/quality/success factors for the product under testing. Analysis of test factors must be performed and then it must be prioritized and ranked. There may be correlation between different test factors. In case of any trade-off, need to be discussed with customer.
2. Each success factor has different importance for different phases. It must be identified to make changes in test approach.
3. The trade off about any test factor must be discussed with customer. Usually trade off leads to risk in development and testing. Impact of test factor is deciding parameter in trade off.
4. Once phase is completed actual impact of risks and effectiveness of solution to risks need to be assessed.

2.25.2 Test Planning

Test plans are different and are based on type of project, product and customer. Following are the various stages of test plan development.

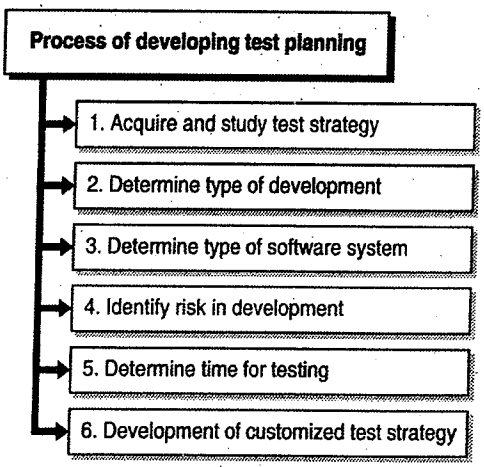


Fig. 2.25.1 : Process of developing test planning

➜ **1. Acquire and study test strategy**

Experienced test team defines the test strategy that focuses testing activity over success factors and risks involved.

➜ **2. Determine type of development**

- Some organizations follow typical waterfall method.
- COTS (Commercial Off the Shelf) purchased software can be integrated with given software system.
- Agile methodology - small iterations and heavy regression testing.
- Spiral development - modular design.

➜ **3. Determine type of software system**

The type of software system being made actually defines the way through which data processing is carried out. It generally involves following points.

- Determine the scope of project.
- Distributed system development is challenging task.
- Integration of various parts developed by different vendors is another challenge.
- While enhancing level of existing system to new one involves defining development and testing scope.
- Change in existing software system involves re-engineering, enhancement, bug-fixing, porting etc.

➜ **4. Identify risk in development**

Risk in a software system arise due to nature of product to be developed, methodology used, skills of teams working on it, type of customer, type of developing organization etc.

Following are some types of risks.

- **Structural risks :** Software design shows need of external libraries and use of complex algorithm will take structure to high risk.
- **Technical risks** : It arise when new technology is used for development that may be new to organization or to the world.
- **Size risks :** Software becomes more complex as it grows in terms of size. Maintaining integrity of such huge size software is a challenge.

➜ **5. Determine time for testing**

- Testing starts from contract, requirement testing till acceptance testing.
- Available testing information is used to determine how much testing is done, where, and when.
- This will define cost of testing, cost of dissatisfaction (customer), and cost of trade-offs.
- Test plan decides various testing activities to be performed.
- It also describes objectives, risks, specific tests, functionality to be tested etc.

➜ **6. Development of customized test strategy**

- The quality factors or test factors need to select and rank as per customer expectations.
- Quality factors are prioritized on a scale of 1-99 where 1 for highest priority and 99 for lowest priority. Same priority will not be allocated for two different factors.
- The phases of system development need to be identified to control these factors.
- Priority of factors may get changed in different phases based on defects originated in respective phase.
- If quality factors are not fulfilled the risk gets introduced in a system.
- These risks are placed in matrix for analysis and deriving preventive, corrective and detective measures to control risk.

Table 2.25.1 : Test strategy matrix

Quality Factors	Test Phase	Requirement	Design	Coding	Testing	Deployment	Maintenance
Factors	Compliance Accuracy Reliability	Risks associated at different phases for different factors					

➜ **7. Types of development methods affecting test plan**

Following section explains various development methods and associated test tactics. It will be different as per the situation, product type and customer.

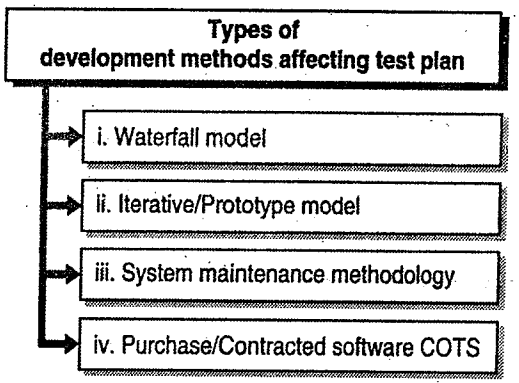


Fig. 2.25.2 : Types of development methods of affecting test plan

➜ (i) Waterfall model

☞ Characteristics

- Uses defined development method.
- Requirements are defined and customer knows it completely.
- Development decides the structure of application.

☞ **Test tactics**

- Testing at the end of each phase to ensure fulfilling of exit criteria.
- Verify match between specification and needs.
- Testing of functions and structure as per test plan.

➜ **(ii) Iterative / Prototype model**

☞ **Characteristics**

- Entire set of requirements unknown to customer as well as to developer.
- Structure is predefined by development model.
- If required, re-factoring is made.

☞ **Test tactics**

- Verify use of case tools in testing.
- Functionality is tested first and then integration, system tests etc.

➜ **(iii) System maintenance methodology**

☞ **Characteristics**

Structure modified if there is limited scope for maintenance activities.

☞ **Test tactics**

- Consistency of structure is tested to avoid wrong impact on entire application.
- Needs heavy regression testing due to continuous system updates.

➜ **(iv) Purchase / Contracted software COTS**

☞ **Characteristics**

- Testers are unknown about system.
- Defects may be hidden.
- User manual contains functionality.
- Variation in document based on type of software product.

☞ **Test tactics**

- Verify match between functionality and business needs.
- Test functionality according to business needs.

Syllabus Topic : Testing Process

2.26 Testing Process

Testing process is comprised of different milestones that needs to be achieved by tester. Milestones depends on organization and project. Every milestones forms strong basis for next stages. Following are the different milestones.

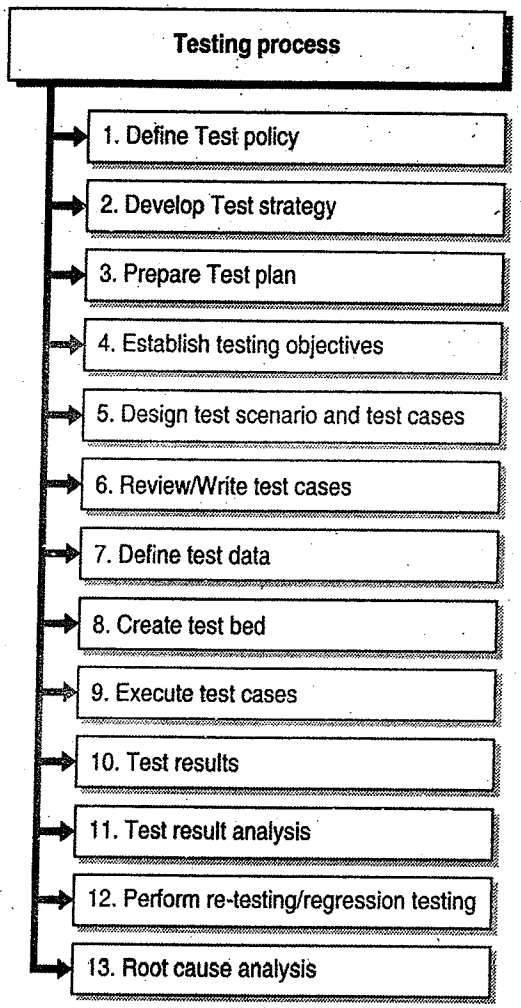


Fig. 2.26.1 : Testing process

➜ **1. Define test policy**

- The test policy is defined by senior management or test management.
- Policy defines intent of organization and is depends on organization maturity, test process, type of customer, software type, development method etc.
- Test policy can be used at project level also based on scope of project.

➜ **2. Develop test strategy**

- The test strategy tells about organization of test team and the way they are going to achieve test objectives, coverage and automation testing.
- Test strategy are the actions to intent defined in test policy. It helps testers to understand testing approach.

3. Prepare test plan

- The test plans are derived from the test strategy. It detail out the way of executing various testing tasks.
- Test plans are defined for project, product or customer. It basically answers six questions as; "what, when, where, why, which and how".
- Generally it is developed by senior tester, test manager or test lead.

4. Establish testing objectives

- Testing objectives should be SMART i.e. Specific, Measurable, Agreed, Realistic, Time-bound. It defines achievement factor.
- Efficiency and effectiveness of testing process can be measured using these objectives. They may be derived using quality objectives.

5. Design test scenario and test cases

- Test strategy defines the way of designing of scenarios and test cases.
- Scenarios comprises actors and transactions and then help to derive test cases.

6. Review / Write test cases

- Test cases are written using test scenarios. Senior tester writes and reviews the test cases.
- Test case and test result are then updated in TM (traceability matrix).

7. Define test data

- Test data is used for executing test cases and it varies according to type of test cases.
- It may be valid or invalid data. It is derived using techniques such as, equivalence partitioning, error guessing, boundary value analysis, state transition etc.

8. Create test bed

- Environment for conducting testing activities is created.
- It may be real-time or simulated environment.

9. Execute test cases

Test data is used to execute test cases in order to get actual result.

10. Test results

- Test log contains all test results. Multiple iteration data is generated.
- Defect repository also gets populated. If actual result does not match with expected result then it must be traced down to requirements.

11. Test result analysis

- Examine the test result to understand strength and weaknesses in software.
- It decides whether to put software in next stage or need to do some rework and re-testing.

12. Perform re-testing/regression testing

- When defects are detected and reported, development team analyze and fix those.
- Re-testing is applied to check whether defects are fixed completely or not.
- Regression testing is carried out to check whether changed part in product has not affected the working of rest of the parts.

13. Root cause analysis

- Weaker areas from development and testing process are noted and root cause analysis is done in order to plan and initiate corrective actions.
- The defect prevention activities are initiated to make improvement in the overall process.

Syllabus Topic : Attitude Towards Testing

2.27 Attitude Towards Testing

- It is very important aspect of senior management, development people, project management towards a testing team. It helps to build morale of testing team.

- It can be initiated from test policy and implemented down-line through test strategy to test planning.
- Development and test team must focus on ultimate objective of achieving customer satisfaction.
- Following are some of the views made about testing team :
  - Defects are taken as personal blames instead of considering it is due to system deficiencies or incapable processes.
  - People always defend themselves, considering blaming as a personal attack.
  - Developers will not accept the defect in first place. They will accept it by putting its blame on someone else.
  - Root cause analysis is very difficult due to involvement of personal egos.
  - Differences are created between development team and testing team due to conflict between developers and testers.

Syllabus Topic : Approaches

2.28 Test Approaches (Methodologies)

1. Black box testing

- It is also known as domain testing or specification testing. It tests the software product as per requirement specification document defined by business analysts and customer.
- It tests the executable system considering inputs, outputs, functionality defined in specification document.
- Testing is carried out independent of database, platform to ensure proper working of system as per specifications.
- Black box testing represents actual user interactions and scenarios and test the behavior of system.
- It is conducted for integration testing, system testing and acceptance testing.

☞ **Advantages**

(i) It proves that system is working properly and performing intended functions.

(ii) Can be used to perform performance testing and security testing.

☞ **Disadvantages**

(i) Coding errors, logical errors cannot be tested.

(ii) System design and internal structure can be missed during testing.

(iii) Redundant testing is carried out. e.g. A function is called and it gets executed again and again due to its placement on same code branch.

☞ **Test case design methods**

- Black box testing does not consider any assumption about how the system is built, hence writing test cases becomes quite difficult.
- The test cases are generated through user scenarios (also known as test scenarios). It contains activities, actions, and transactions of users.
- Test data being used for testing should be well enough to produce adequate coverage, sufficient test cases, and avoids system failure. Following are the black box test case design methods :
  (i) Equivalence partitioning
  (ii) Boundary value analysis
  (iii) Cause and effect graph
  (iv) State transition testing
  (v) Use case based testing
  (vi) Error guessing

2. White box testing

- It is focused on internal structure, architecture, coding standards, guidelines of software.
- White box testing ensures the correctness in relationship between requirements, design, and coding of software.
- It is a verification technique to ensure whether software is built correctly or not.

☞ **Advantages**

(i) It is the only way to check whether documented procedures, processes, standards, and methods are really followed or not during development.

(ii) It ensures coding standard, reuse followed or not.

(iii) If something is not done properly then it can be indicated and detected early.

(iv) Some parts are verified only such as code complexity, commenting, reuse etc., and is possible due to white box testing.

☞ **Disadvantages**

(i) No assurance whether customer requirements are met correctly.

(ii) Code is not executed fully, hence no guarantee about its proper working.

☞ **Test case design methods**

(i) Statement coverage

(ii) Decision coverage

(iii) Code coverage

(iv) Path coverage

(v) Condition coverage

3. Gray box testing

- It is done on the basis of internal structure, requirements, design, coding standards, guidelines along with functional and non-functional requirements specifications.
- It combines both, verification and validation techniques.

☞ **Advantages**

(i) It helps to ensure correctness of software both structurally and functionally.

(ii) Combines advantages of both black box and white box testing methods.

☞ **Disadvantages**

(i) Need to be conducted using tools.

(ii) Knowledge of usage of tool and configuration of tool is required.

Syllabus Topic : Challenges

2.29 People Challenges in Software Testing

- Tester is responsible for testing process improvement that will ensure quality and less defects in product.
- This will increase customer satisfaction level. Proper coverage as per test plan must be achieved.
- Tester must have all required skills and training essential to deliver defect free product to customer.
- Testers need to improve their skill through continuous learning process.
- Tester must possess positive team attitude. He should look at the found defects as an opportunity instead of blaming developers.
- He must point out flaws in development process.
- Testing can be performed effectively through creative design of test scenarios and test cases.
- Tester and developer must work hand in hand by keeping common objective of achieving customer satisfaction.
- Both must try to improve quality of product and development process.
- Tester must close the defect by conducting retesting and regression testing. He must also check for any negative impact of fixed defect on changed product.

Syllabus Topic : Raising Management Awareness for Testing

2.30 Raising Management Awareness for Testing

In every organization, management must be aware about roles and responsibilities of test team personal, finding defects and achieving customer satisfaction through defect-free product delivery. By finding all possible defects, testers are contributing in building quality product.

Role of tester

- Tester must understand the objectives set by management.
- Calculate cost and effectiveness of testing and communicate same to management.
- Perform good testing so that it will reduce customer complaints and failure cost.
- Highlight the need of training on certain skills required by developers and testers. It will ensure more better performance by people involved.
- Convey importance of unit, integration testing to development team.
- Collect and distribute all testing information to all team members, management.
- Develop testing budget that focuses on people, money, time, training etc.

Syllabus Topic : Skills Required by Tester

2.31 Skills Required by Tester

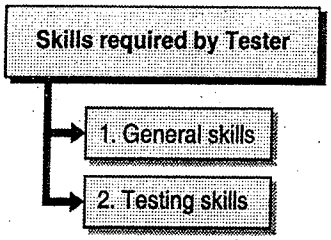


Fig. 2.31.1 : Skills required by tester

1. General skills

(i) Verbal presentation and written skill : test result presentation, defect communication.

(ii) Listening skill : Need to listen to developer, sometimes customers.

(iii) Facilitation skill : facilitation of development team and customers is done by test team.

(iv) Development, maintenance, and operation : good knowledge and understanding of SDLC and STLC (Software Testing Life Cycle).

(v) Continuous Education : Tester need to be updated with training programs.

2. Testing skills

(i) Testing fundamentals

(ii) Levels in Testing

(iii) Verification and Validation techniques

(iv) Selecting and using test tools

(v) Testing standards

(vi) Risk assessment and management

(vii) Test plan development

(viii) Definition of acceptance criteria

(ix) Test process assessment

(x) Test plan execution

(xi) Test process improvement

Review Questions

Q. Explain the evolution of software testing from debugging to prevention based testing. **(Section 2.1)**

Q. Why independent testing is necessary ? **(Section 2.1.2)**

Q. Explain big bang approach of software testing. **(Section 2.1.3(A))**

Q. Explain TQM approach of software testing. **(Section 2.1.3(B)**

Q. What do you mean by TQM cost perspective ? **(Section 2.1.3(B)(ii)**

Q. Explain testing as a process of software certification. **(Section 2.1.3(D))**

Q. Explain basic principles on which testing is based. **(Section 2.1.4(G))**

Q. Explain the concept of test team's defect finding efficiency. **(Section 2.14)**

Q. Explain the test case's defect finding efficiency. **(Section 2.15)**

Q. What are the various challenges can be faced during testing ? **(Section 2.16)**

Q. Explain the process of developing test strategy. **(Section 2.25.1)**

Q. Explain the process of developing test planning **(Section 2.25.2)**

Q. Which skills are expected in a good tester ? **(Section 2.31)**

Q. What do you mean by following software testing terms ; defect, error and mistake ? **(Section 2.24)**

Q. Explain in brief three test approaches **(Section 2.28)**

Q. Which are the different categories of defects ? **(Section 2.23)**

Q. Discuss cost aspect in software testing. **(Section 2.19)**

Q. Discuss test team approach along with types. **(Section 2.17)**

Q. Write different misconception in software testing. **(Section 2.7)**

Q. Explain typical tester workbench with suitable diagram. **(Section 2.5)**

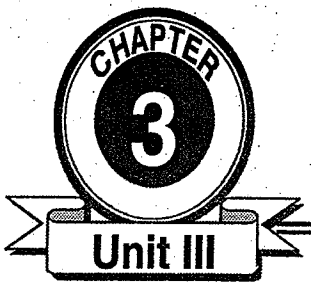
Q. Discuss about SWOT analysis of software testing process. **(Section 2.4)**

Q. Define requirement traceability matrix. **(Section 2.3)**

Q. Explain in brief four types in RTM. **(Section 2.3)**

Q. Write a note on : Testing during development life cycle. **(Section 2.2)**

❑❑❑



Software Test Automation

Syllabus Topics

What is Test Automation, Terms used in automation, Skills needed for automation, What to automate, scope of automation, Design and Architecture of automation, Generic requirement for Test Tool, Process Model for Automation, Selecting Test Tool, Automation for XP/Agile model, Challenges in Automation, Data-driven Testing. Automation Tools like JUnit, JMeter.

Syllabus Topic : What is Test Automation ?

3.1 What is Test Automation ?

- In software testing, test automation is defined as - developing a special software to test the software.
- This special software controls the execution of tests, does the comparison of actual outcomes with expected outcomes etc.
- Test automation is used to perform complex tests that are difficult to perform in a manual way.
- It can be used to automate some necessary tasks in a repetitive way. Widely used automation testing tools are HP QTP (Quick Test Professional)/UFT(Unified Functional Testing), Selenium, LoadRunner, IBM Rational Functional Tester, SilkTest, TestComplete, WinRunner, WATIR etc.

3.1.1 Benefits of Test Automation

- It avoids chance of human error or a bias because of which some of the defects may get missed out.
- It saves time as software can execute test cases faster than human being.
- Automated tests can be run overnight, saving the elapsed time for testing, thereby enabling the product to be released frequently.
- It can free the test engineers from doing unnecessary tasks and make them focus on more creative and useful tasks.
- Automated tests can be more reliable.
- It helps in immediate testing; need not wait for the availability of test engineers.
- It can protect an organization against a gradual reduction in the number of test engineers.
- It can also be used as a knowledge transfer tool to train test engineers on the product as it has a repository of different tests for the product.
- It opens up opportunities for better utilization of global resources.
- It enable teams in different parts of the world, in different time zones, to monitor and control the tests.
- Certain types of testing (like, reliability testing, stress testing, load and performance testing) solely depends on automation.
- It is not limited to test execution but can consider all activities such as picking up the right product build, choosing the right configuration, performing installation, running the tests, generating the right test data, analyzing the results, and filing the defects in the defect repository.

3.1.2 Difference between Automation Testing and Manual Testing

Table 3.1.1 : Difference between Automation Testing and Manual testing

Sr. No.	Automation Testing	Manual Testing
1.	Automation testing perform the repetition of same operation each time.	Manual testing is not reliable since result of test execution is not accurate all the time.
2.	In case of regression testing where codes are changed frequently, the automation testing is very much helpful.	It is difficult to catch defects after regression testing using manual testing.
3.	It is useful when set of test cases needs to execute frequently.	It is useful when the test case needs to run once or twice.
4.	Very few test people are required to execute test cases once automation test suites are ready.	Same amount of time is required by test people to execute the test cases.
5.	It works with heterogeneous environment with different machine, OS, and platform combination, concurrently.	It is impossible to carry out manual testingon different machine, OS, and platform combination, concurrently. Using different test teams such task can be executed.
6.	Testers can test complex applications using automation testing.	Manual testing does not involve any programming task to retrieve hidden information.
7.	Automation process runs test cases faster than human.	Manual testing is slower process than automation. Manual running of test cases can be very time consuming.
8.	In some cases, it is not helpful in testing UI.	It is very much helpful in testing UI.
9.	Automating the Build Verification Testing (BVT) is not unexciting and tiresome.	Executing the Build Verification Testing (BVT) is very unexciting and tiresome.
10.	Initial cost is more than manual testing, but it is always useful.	Manual testing requires less cost than automation testing.

Syllabus Topic : Terms used in Test Automation

3.2 Terms used in Test Automation

Fig. 3.2.1 represents general framework for automation. Terms associated with this framework are explained below.

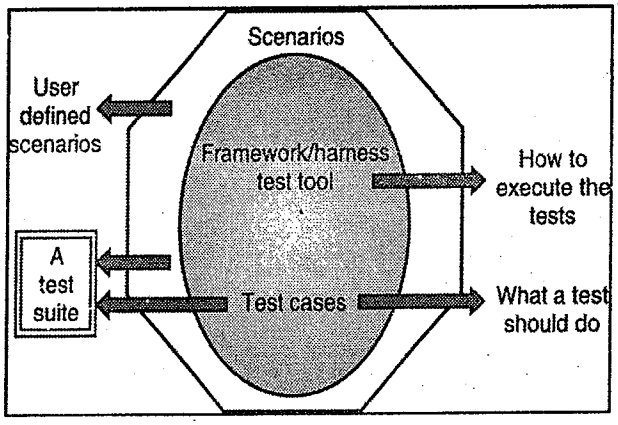


Fig. 3.2.1 : Test automation framework

- **Test Case :** It is a created as a document that consists test data or test input, preconditions, expected results and post conditions. Test cases are developed for a particular test scenario to verify a specific requirement. It is of two types, manual test cases and automated test case.
- **Test Suite :** Test suite is a container that consists a set of tests or test cases.
- **Test Scenarios :** It represents, "how" a test will be carried out or how to execute a particular test case.

Syllabus Topic : Skills Needed for Automation

3.3 Skills Needed for Automation

- There are different "Generations of Automation" such as; first, second and third.
- The skills needed for automation depends on what generation of automation the company is in or desires to be in the near future.

Automation is categorized into three types which are as follows :

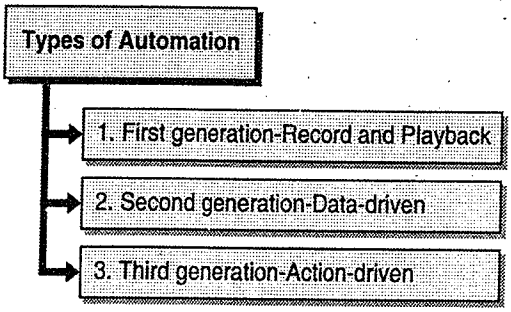


Fig. 3.3.1

1. First generation-Record and Playback

- First generation record and playback avoids the repetitive nature of executing tests.
- A test engineer uses keyboard characters or mouse clicks and records the sequence of actions and those recorded scripts are played back later, in the same order as they were recorded.
- Since a recorded script can be played back multiple times, it reduces the task of testing function. It is simple to record and save the script.

Disadvantages

- The scripts may contain hard-coded values, that makes general types of tests little difficult to perform.
- The task of handling error condition is allocated to testers, this causes a lot of manual intervention to detect and correct error conditions in played back scripts.
- When the application changes, all the scripts have to be re-recorded, thereby increasing the test maintenance costs and automation becomes ineffective.

2. Second generation-Data-driven

- It focuses on input and output conditions using the black box testing approach.
- The test scripts that generates the set of input conditions and corresponding expected output are developed using this method.
- Because of that, tests can be repeated for different input and output conditions.
- However, changes to application does not require the automated test cases to be changed as long as the input conditions and expected output are still valid.

3. Third generation-Action-driven

- In this method, no input and expected output conditions required for running the tests.
- All actions that appear on the application are automatically tested, using a generic set of controls defined for automation.
- The user needs to specify only the operations; for example, log in, download etc. and everything else that is needed for those actions are automatically generated.
- The input and output conditions are automatically generated and used. Two major aspects – "test case automation" and "framework design" are involved in third generation.
- The skills needed for automation are classified into four levels for three generations as the third generation of automation introduces two levels of skills for development of test cases and framework, as shown in Table 3.3.1.

Table 3.3.1 : Classification of skills for automation

First Generation	Second Generation	Third Generation	
Skill for test case automation	Skill for test case automation	Skill for test case automation	Skill for framework
Scripting languages	Scripting languages	Scripting languages	Programming languages
Record-Playback tools usage	Programming languages	Programming languages	Design and architecture skills for the framework creation
	Knowledge of data generation techniques	Design and architecture of the product under test	Generic test requirements for multiple products
	Usage of the product under test	Usage of the framework	

Syllabus Topic : What to Automate ?, Scope of Automation

3.4 What to Automate ?, Scope of Automation

Following are the various ways to identify scope of automation testing.

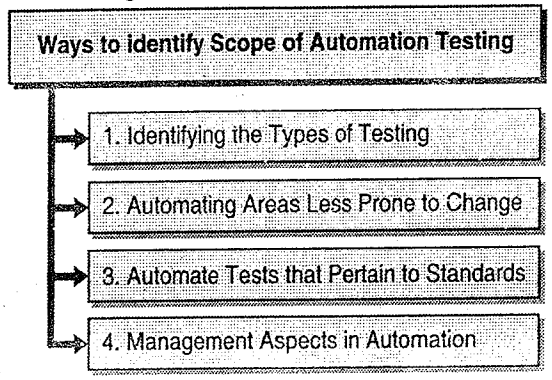


Fig. 3.4.1

➜ **1. Identifying the Types of Testing**

- Test cases belonging to testing types such as, Stress, reliability, scalability, and performance testing are suitable for automation.
- The test cases are executed for longer time, and in a repetition. Automation will save significant time and effort in the long run of regression test cases.
- Functional tests can be automated, which can done by less-skilled people.

➜ **2. Automating Areas Less Prone to Change**

- Areas where requirements are not changed are considered for automation.
- In order to avoid rework on test cases in automation testing, proper analysis must be done to find out the areas of changes to user interfaces, and automate only those areas that will go through relatively less change.

➜ **3. Automate Tests that Pertain to Standards**

- The products are tested for compliance to standards. Test suites developed for standards are used for product testing, and can be sold as test tools for the market.
- Certification testing suites used for testing of software and hardware before their release are considered for automation test.

➜ **4. Management Aspects in Automation**

- Automation effort should focus on those areas for which management commitment exists already. Another management aspect is Return on investment.
- Clear estimates must be given for efforts and time needed for automation. Automate the critical and basic functionality of a product first. Focus on test cases with less code.
- The test cases which are easy to automate in less time should be considered first for automation.
- Automation should start from high priority and then cover medium and low-priority requirements.

Syllabus Topic : Design and Architecture for Automation

3.5 Design and Architecture for Automation

- Architecture of automation includes two major things, a test infrastructure that covers a test case database and a defect database or defect repository.
- External interface shown by thick arrows whereas thin arrows shows internal interfaces. It is shown with the help of Fig. 3.5.1.
- Using this infrastructure, the test framework provides a backbone that ties the selection and execution of test cases.

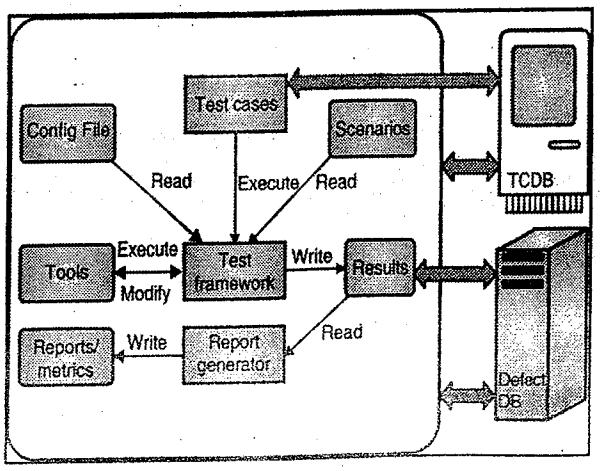


Fig. 3.5.1 : Components of test automation

1. External Modules

- TCDB (test case DB) and defect DB are the two external modules. Any module from framework can access these two external modules.
- TCDB stores, all the test cases (both, manual and automated), the steps to execute them, and the history of their execution (such as when a particular test case was run and whether it passed/failed).
- The interaction between TCDB and the automation framework only for automated test cases represented by interface with thick arrow.
- Defect DB or defect database or defect repository contains details of all the defects (such as, when the defect was found, to whom it is assigned, what is the current status, the type of defect, its impact, etc.) that are found in various products that are tested in a particular organization.
- Defects for manual test are submitted by test engineer whereas for automation test, it is submitted by framework automatically to Defect DB.

2. Scenario and Configuration File Modules

- A configuration file contains a set of variables that are used in automation.
- The variables are related to test framework, tools and metrics, test suite, set of test cases, particular test case etc.
- A configuration file is important for running the test cases for various execution conditions, running the tests for various input and output conditions and states.
- By changing the values of variables in this configuration file dynamically, different execution, input, output, and state conditions can be achieved.

3. Test Cases and Test Framework Modules

- Automated test cases that are taken from TCDB and executed by the framework. Other modules in the architecture are using test case as an object.
- Test framework is a core part of this framework and one can develop or purchase it.
- It picks up the specific test cases that are automated from TCDB and picks up the scenarios and executes them.
- It picks up variables along with their defined values and executes test cases with them.
- It subjects the test cases to different scenarios. It handles interaction, initiation, and controlling of all modules.

4. Tools and Results Modules

- Set of run-time tools and utilities are required by test framework when performing certain operations like executing test cases.
- Result of each set of test case execution is preserved and used for future analysis and action.
- The current results should not overwrite the results from the previous test runs.
- Archives are created with the history of all the previous tests run that will help in executing test cases based on previous test results.

5. Report Generator and Reports/Metrics Modules

- Test reports and metrics are prepared (daily, weekly, monthly, or milestone) once test run is over and results are available.
- Report generation is a part of automated design which should generates some customized reports viz.
  - **Executive report :** Gives very high level status;
  - **Technical reports :** Give a moderate level of detail of the tests run.
  - **Detailed or debug reports :** Generated for developers to debug the failed test cases and the product.
- Report generator module takes the necessary inputs and prepares a formatted report.
- It then generates metrics based on available results. Reports/metrics module acts as storage and keeps all generated reports and metrics that can be used for future use and analysis.

Syllabus Topic : Generic Requirements for a Test Tool

3.6 Generic Requirements for a Test Tool / Framework

This section will present certain detailed criteria that a framework and its usage should satisfy.

Requirement 1 : No hard coding in the test suite.

- Keep all configuration variables in configuration file. Variables belonging to the test tool and the test suite need to be separated.
- Providing inline comment for each of the variables will make the test suite more usable and may avoid improper usage of variables.

Requirement 2 : Test case/suite expandability.

Requirement 3 : Reuse of code for different types of testing, test cases.

Different scenarios becomes test cases for different types of testing. This encourages the reuse of code in automation. The test programs need to be modular to encourage reuse of code.

Requirement 4 : Automatic setup and cleanup.

- Each test program should have a "setup" program that will create the necessary setup before executing the test cases.
- The test framework should find out intelligently what test cases are executed and call the appropriate setup program.
- It is also important to have cleanup program after test execution for a test case is over. This will reset the automation.

Requirement 5 : Independent test cases.

Requirement 6 : Test case dependency.

- Independence will enable test engineer to select any one case at random and execute it.
- There may be a dependent test case which need information to execute before or after a dependent test case is selected for execution.
- A test tool or a framework should provide both features and help to specify the dynamic dependencies between test cases.

Requirement 7 : Insulating test cases during execution.

- During the test case execution, there could be some events or interrupts or signals in the system that may affect the execution.
- The framework should provide an option for users to block some of the events.

Requirement 8 : Coding standards and directory structure.

- This will help the new engineers in understanding the test suite fast and help in maintaining the test suite.
- The test framework should provide an option for directory structure to enable parallel test case development, without duplicating the parts of the test case and by reusing the portion of the code.

Requirement 9 : Selective execution of test cases.

The test framework should have a facility for the test engineer to select a particular test case or a set of test cases and execute them.

Requirement 10 : Random execution of test cases.

- A test engineer may sometimes need to select a test case randomly from a list of test cases.
- A test engineer selects a set of test cases from a test suite; selecting a random test case from the given list is done by the test tool called as random execution of test cases.

Requirement 11 : Parallel execution of test cases.

- Parallel tests can run in a multi-tasking and multi processing operating systems.
- Parallel execution simulates the behavior of several machines running the same test and hence is very useful for performance and load testing.

Requirement 12 : Looping the test cases.

- Reliability testing type executes the test cases in a loop.
- Two types of loops are there, iteration loop which gives the number of iterations of a particular test case to be executed and the timed loop, which keeps executing the test cases in a loop till the specified time duration is reached.

Requirement 13 : Grouping of test scenarios.

- It is useful to combine individual scenarios into a group, which can run for a long time with a good mix of test cases.
- The group scenarios allow the selected test cases to be executed in order, random, in a loop all at the same time, or in a predetermined combination of scenarios.

Requirement 14 : Test case execution based on previous results.

In regression testing and some other types, requires that test cases be selected based on previous result and history.

Requirement 15 : Remote execution of test cases.

- Many software products need to be tested using more than one machine. In such case the central machine that allocates tests to multiple machines and coordinates the execution, tracks the test progress, and collects the result is called test console or test monitor.

Requirement 16 : Automatic archival of test data.

- All the related information for the test cases have to be archived for repeating test cases and for analysis purpose.
- It helps in knowing information like, configuration variables used, scenario used, and programs executed with their path.

Requirement 17 : Reporting scheme.

- All the test suite needs to have a reporting scheme (similar to report generator from design and architecture of framework) from where meaningful reports can be extracted.
- Report must tell information like; scenario, test suite, test program, and each test case status, Result of each test case, Log messages, Category of events and log of events and Audit reports.

Requirement 18 : Independent of languages.

The test framework/test tool

- should be independent of programming languages and scripts.
- should provide a choice of languages, scripts, and their combination that are popular and useful in the software development area.
- should work with different test programs written using different languages and scripts.
- should have interfaces exported to support all popular, standard languages, and scripts.

Requirement 19 : Portability to different platforms.

- Test tools and framework must be a cross-platform, supporting platform-independent languages, technologies, and diversified environments.
- Portability and compatibility must be supported.

Syllabus Topic : Process Model for Automation

3.7 Process Model for Automation

- Automation testing can be started and finished at any time irrespective of any rule. Product development and automation can go simultaneously.
- It is also referred as two parallel tracks of a railway line. The results of automation must be obtained before test execution so that automation efforts can be utilized for the current product release. After product release, test execution may be stopped but automation can continue.
- Based on the similarities between automation development and product development, the process and life cycle model to be followed also can be similar.
- The V-model can be extended to show process of test automation. Fig. 3.7.1 shows phases involved in product and automation development.
- Similarities can be - product requirement gathering and automation requirement gathering; product planning and automation planning etc.
- It is very important to test the test suite, because defective test suite leads to have more efforts during debugging to determine whether it is from product or test suite itself.

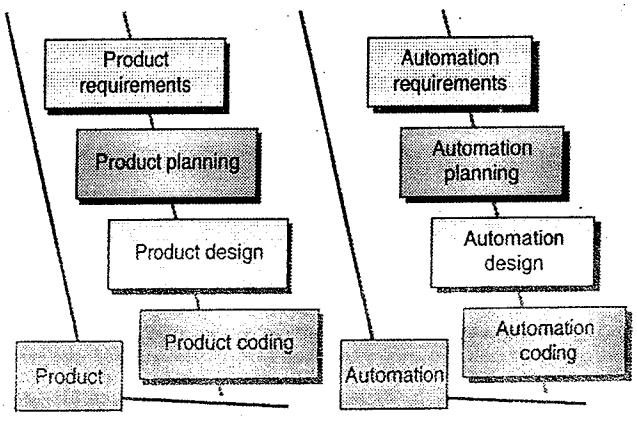


Fig. 3.7.1 : Product development and Automation

- During automation also, test suite of automation testing need to be tested first. If not, it will affect on credibility and faith of automation.
- So it is necessary to provide quality test suite to the automation in order to generate quality product. This can be supported by imparting test suite testing phase for product and automation.
- At each phase of product and automation development, a set of four activities can be performed.
- While collecting development requirements, test requirements for product; simultaneously these can be taken for automation also (i.e. Requirements for automation development and automation testing).
- These four activities can be performed at planning and design phases.
- The coding phase of product and automation gives coding phase in below W-model (Fig. 3.7.2) where product and test suites are delivered.
- The deliverables from automation cane be test suite and test framework.
- Fig. 3.7.2 represents two parallel set of activities for development and two parallel set of activities for testing.
- By combining them together produces a W-model. This will ensure good quality test suite and product.
- The activities from particular phase shown in W-model need not to start and finish at same time, e.g. 'Design', 'test design', 'automation design' and 'test design for test suite' appear in same phase, shown side by side.
- Start and end dates for these activities for product and automation can be different.
- Flow of activities is the main concern not the schedule. Product development and automation can be performed independently.
- This approach is followed mostly because many organizations does not have dedicated test team for automation.
- So managing schedule is difficult for them .the start and end dates are decided based on availability of resources, schedule and other dependencies.

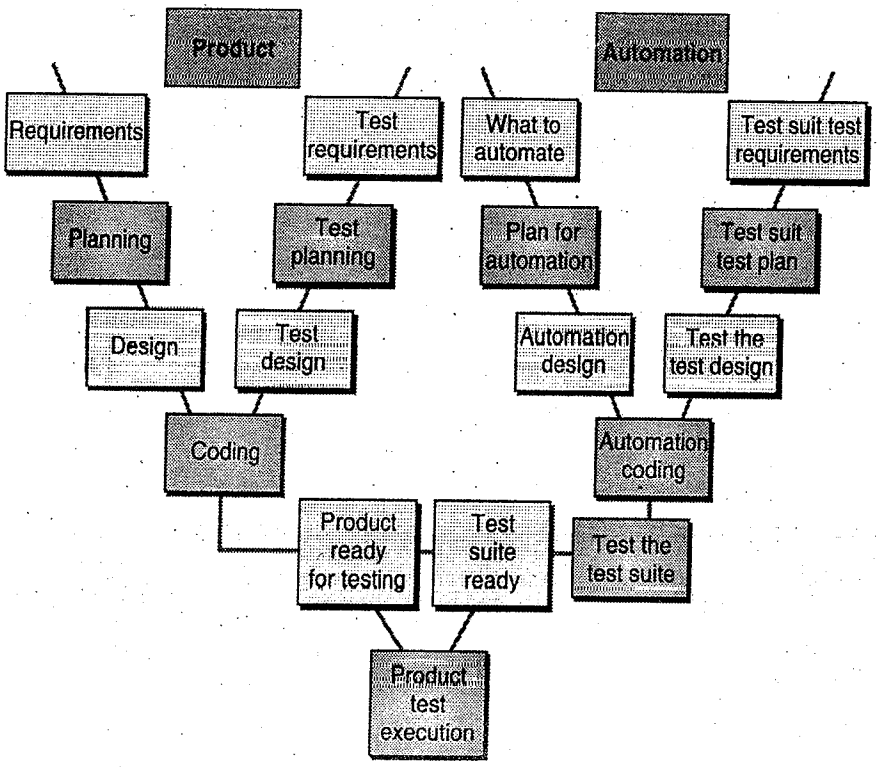


Fig. 3.7.2 : W-Model

Syllabus Topic : Selecting Test Tool

3.8 Selecting Test Tool

The requirement for automation are somewhat dependent of test tools. These requirements are short term and long term and it forms the basis for selecting test tool.

3.8.1 Why Selecting Test Tool is Important?

1. Free tools are having less support and can delay release.
2. In-house development of tool needs more time. It is less expensive but may be poorly documented. If originator of tool leaves organization it becomes unusable. It is not able to sustain pressure of schedule.
3. Standard tools from vendors are expensive and small/medium organization need evaluate economic impact.
4. Training is an important consideration. Automation will be successful if testers are well trained and skilled. Training generally includes, scripting language, tool customization, adding extensions and plug-ins, which will require high amount of efforts.
5. Customization and extensibility of a test tool is important issue.
6. Portability of tests (includes scripts) and tool on multiple platform is a major issue.

3.8.2 Criteria for selecting Test Tool

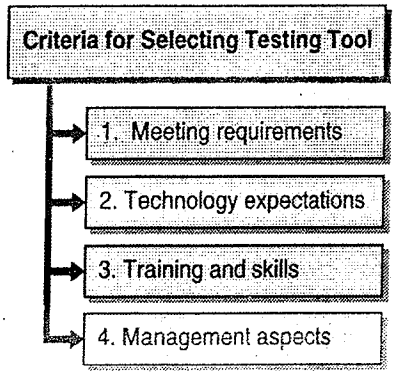


Fig. 3.8.1 : Criteria for Selecting Testing Tool

➜ **1. Meeting requirements :** Not all tool fulfill every requirement, thus evaluating tools for requirements need time, efforts and money. This will cause delay in deciding and implementing tool. Some tools are not having backward and forward compatibility. In case of new requirement, test tools are not going in same level of details. Many tools are not able to differentiate between product failure and test failure. This requires analysis through debugging and troubleshooting. It may require manual testing.

➜ **2. Technology expectations :** Important expectations of a test tool is extensibility and customization. Tools are not allowing test developers to extend or modify functionality of framework. To achieve this, extra cost and efforts are required. There are number of tools in market which requires linking of product binary with tool libraries. An instrumental code (source code plus tool libraries) is produced. Testing is repeated after removing libraries. Few types of testing performs better when such libraries are removed. e.g. Performance testing is impacted due to instrumented code, since extra code requires extra time in executing. Many tools are not supporting cross-platform feature hence script developed/executed on one platform are not functioning on another.

➜ **3. Training and skills :** Many tools requires training and it becomes a new learning curve for tester. They have to learn scripting language. Organization provide training for deployment of tools.

➜ **4. Management aspects :** Test tool always increases the system requirements in terms of software and hardware. Many times tools requires upgrade in existing software and hardware which involves very high cost. Shifting from one tool to another may cause non functioning of scripts. Management must feel returns on investment then only migration of tools can be permitted. Deploying tools in an organization also very time consuming activity and needs more time and efforts. This can cause delay in projects.

Table 3.8.1 : Issues in selecting a test tool

Meeting requirements	Technology expectations	Training / skills	Management aspect
Checking whether the tools meet requirements involves effort and money.	Extending the test tool is difficult.	Lack of trainers for test tools.	Test tools require system upgrades.
Test tools are not fully compatible with products.	Requires instrumented code to be removed for certain tests.	Test tools require people to learn new language/scripts.	Migration to other test tools difficult.
Test tools are not rested with the same seriousness as products for new requirements.	Test tools are not cross-platform.		Deploying tool requires huge planning and effort.
Difficult to isolate problems of product and test suite; change in product causes test suite to be changed.			

Syllabus Topic : Automation for XP/Agile Model

3.9 Automation for XP

- Automation is an integral part of product development in extreme programming model. It is not considered as an extra activity.
- Automation requirements are collected at the time of product requirements only. This indicates that automation should be started from the first day of product development.
- In extreme programming test cases are written before coding phase starts. Developers will develop a code to ensure passing of test cases.
- This will form a sync between code and test cases. The code written for product can be reused during automation. Developer automatically develops test cases for automation since objective of code is to ensure passing of test cases.
- All unit test cases should be executed with 100% pass rate in extreme programming.
- There will not be any gap left between development and coding skills for automation team in extreme programming.
- People cross boundaries to perform different roles as developer, testers etc. Extreme programming puts development, testing and automation skills at single place.

Syllabus Topic : Challenges in Automation

3.10 Challenges in Automation

- Automation should not be viewed as a solution or quick-fix for all the quality problems in a product.
- It is required to invest significant amount at the beginning. Test engineers need to follow wide and steep learning curve.
- Management should have patience and persist with automation and should not start looking for an early payback,
- Success of automation depends on fearless management, a clear vision of the goals, and the ability to set realistic short-term goals that track progress with respect to the long-term vision.

Syllabus Topic : Data Driven Testing

3.11 Data Driven Testing

- Data Driven Testing (DDT) is a software testing methodology that is used in the testing of computer software.
- Data driven testing is where the test input and the expected output results are stored in a separate data file i.e. normally in

a tabular format, so that a single driver script can execute all the test cases with multiple sets of data.

- It can be any tabular format file or data file
  - Datapools
  - Excel files
  - ADO objects
  - CSV files
  - ODBC sources
- This can allows test automation engineers to have a single test script which can execute tests for all the test data in the table.

3.11.1 Data-Driven Test Automation Framework

- Data-driven test automation framework shown in Fig. 3.11.1.

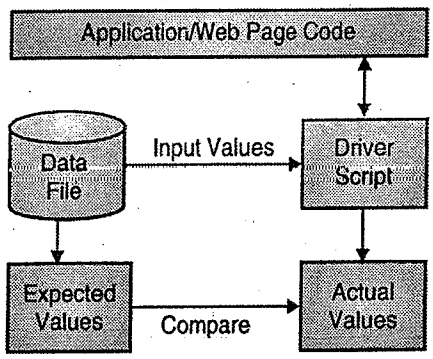


Fig. 3.11.1 : Data Driven Testing Framework

- In this framework, input values are read from data files and are stored into a variable in test scripts.
- It enables building both positive and negative test cases into a single test.
- In Data-driven test automation framework, input data can be stored in single or multiple data sources like xls, XML, csv, and databases.
- Many times, there is a requirement to run same test on different types of data set. For this purpose tester writes different test for each data set.
- It is time consuming and inefficient.
- Solution for this problem is Data Driven Testing. This overcomes this issue by keeping the data external to Functional tests and loading them when there is a need to extend the automation tests.

Example : Write data driven testing steps for login page of the flight reservation.

1. The first step involves creating the test data file. This data file contains the different types of input data which will be given to the driver script.

Test Case	Number1	Operator	Number2	Expc. Result
Add	2	+	3	5
Subtract	3	–	1	2
Multiply	2	*	4	8
Divide	2	/	–2	–2

2. This data file contains the different types of input data which will be given to the driver script.

```
data = open ( ' testdata.csv' ). read ( )
lines = data.splitlines( ) #excluding the header row
for line in lines :
    Read Number1
    Read Number2
    Read Operator
    Calculate the result using the Operator on
    Number 1 and Number2
    Compare the result to the expected result
```

This driver script reads the data from the data file computes the value and compares it with the expected result from the data file.

Advantages

- This framework reduces the number of overall test scripts needed to implement all the test cases.
- Less amount of code is required to generate all the test cases.
- Offers greater flexibility when it comes to maintenance and fixing of bugs.
- The test data can be created before test implementation is ready or even before the system to be tested is ready.

Disadvantages

- The test cases created are similar and creating new kind of tests requires creating new driver scripts that understand different data. Thus the test data and driver scripts are strongly related that changing either requires changing the other.

Syllabus Topic : Automation Tools - JUnit

3.12 JUnit

- Test Driven Development (TDD) is one of the most successful productivity enhancing techniques used by many in eXtreme Programming. TDD has three step: write test, write code & refactor.
- JUnit is an open source Java testing framework used in TDD. JUnit was originally written by Erich Gamma and Kent Beck. It is an instance of the xUnit architecture for unit testing frameworks.

3.12.1 Why use JUnit ?

- It is open source means free!
- It is simple and elegant to use.
- It is easy and inexpensive to write tests using the JUnit testing framework.
- JUnit tests checks their own result and provide quick visual feedback.
- Tests can be composed into TestSuites.
- It is integrated into IDE's like Eclipse and NetBeans.

3.12.2 Design of JUnit

- JUnit is designed around two key design patterns :
  - The Command pattern
  - The Composite pattern.
- A TestCase is a Command object. It can define any number of public testXXX() methods and any test class that has test methods should extend the TestCase class.
- The test class can use the setUp() and tearDown() methods to initialize and release common objects under test, referred to as the test fixture.
- TestCase instances can be composed into TestSuites that can automatically invoke all the testXXX() in each TestCase instance.
- TestSuite can also have other TestSuite instances and this allows the test to run automatically and provide test status.

3.12.3 How to install JUnit?

- First, download the latest version of JUnit from http://www.junit.org (Eclipse should already have JUnit installed.)
- For installing JUnit on Windows :
  - Unzip the junit.zip to %JUNIT_HOME%
  - Add JUnit to classpath:
  - set CLASSPATH=%JUNIT_HOME%\junit.jar
- For installing JUnit on UNIX :
  - Unzip the junit.zip to $JUNIT_HOME
  - Add JUnit to classpath:
  - export CLASSPATH=$JUNIT_HOME/junit.jar
- Test installation by using the textual or graphical test runner to run the sample tests distributed with JUnit.
- All tests should pass with an "OK" or a green bar. If not, then verify that junit.jar is in the CLASSPATH.

3.12.4 JUnit Test Case

To write a test case follows these steps :.

- Define a subclass of TestCase.
- Overide the setUp() method to initialize objects under test.
- Optionally overide tearDown() method to release object under test.
- Define one or more public testXXX() methods that exercise the objects under test.

☞ **Sample Test Case**

```
import junit.framework.TestCase;
public class ShoppingCartTest extends TestCase {
        private ShoppingCart cart;
        private Product book1;
        // Sets up the test fixture.
        //Called before every testcase method.
            protected void setUp() {
            cart = new ShoppingCart();
book1 = new Product("Pragmatic Unit Testing", 29.95);
            cart.addItem(book1);
        }
        //Tears down the test fixture.
        //Called after every test case method.
        protected void tearDown() {
            // release objects under test here
            }
            //Tests emptying the cart.
public void testEmpty() {
                cart.empty();
                assertEquals(0, cart.getItemCount());
            }
...
};
```

3.12.5 JUnit Test Suite

- A TestSuite comprises of various TestCase instances.
- To write a test suite follow these steps :
  - Create a class that defines a static suite() factory method which creates TestSuites for all tests.
  - Optionally define a main() method that runs the TestSuite in batch mode.

☞ **Sample Test Suite**

```
import junit.framework.Test;
import junit.framework.TestSuite;
public class EcommerceTestSuite {

  public static Test suite() {
     TestSuite suite = new TestSuite();
     // The ShoppingCartTest we created above.
     suite.addTestSuite(ShoppingCartTest.class)
  // Another example test suite of tests.
     suite.addTest(CreditCardTestSuite.suite());
     // Add more tests here
     return suite;
  }
  //Runs the test suite using the textual runner.
  public static void main(String[] args) {
     junit.textui.TestRunner.run(suite());
  }
```

3.12.6 How to Run JUnit Tests?

- JUnit provides textual and Graphical Interface (GI) to run tests. Both interface indicates how many tests were run, errors or failures and test status, "OK" for textual and green bar in GI.
- For running tests from textual interface : java junit.textui.TestRunner ShoppingCartTest
- For running tests from graphical interface : java junit.swingui.TestRunner ShoppingCartTest

3.12.7 Assertions for JUnit

JUnit uses assertion methods to test conditions :

- assertEquals(a,b) - a and b must be primitives or have an equals method for comparison.
- assertFalse(a) - a must be boolean.
- assertNotNull(a) - a is either object or null.
- assertNotSame(a,b) - test for object equality.
- assertNull(a) - a is either object or null.

- assertSame(a,b) – test for object equality.
- assertTrue(a) - a must be boolean.

Syllabus Topic : Automation Tools - JMeter

3.13 JMeter

- **JMeter** is an Apache Jakarta project that can be used load test, performance test, regression test, etc., on different protocols or technologies.
- The protocols supported by JMeter are –
  - Web – HTTP, HTTPS sites 'web 1.0' web 2.0 (ajax, flex and flex-ws-amf)
  - Web Services – SOAP / XML-RPC
  - Database via JDBC drivers
  - Directory – LDAP
  - Messaging Oriented service via JMS
  - Service – POP3, IMAP, SMTP
  - FTP Service
- JMeter can be used as a unit test tool for JDBC database connection, FTP, LDAP, WebServices, J MS, HTTP and generic TCP connections.
- A GUI desktop application Built in Java; designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.
- JMeter can also be configured as a monitor, although this is typically considered an ad-hoc solution in lieu of advanced monitoring solutions.
- **JMeter is not a browser** - JMeter does not execute the Javascript found in HTML pages. Nor does it render the HTML pages as a browser does, but it is possible to view the response as HTML etc, but the timings are not included in any samples, and only one sample in one thread is ever viewed at a time.
- JMeter works at protocol level.
- It can run on any workstation or environment that can run java.
- It has a GUI and non-GUI mode
  - GUI for developing and watching tests.
  - non-GUI for running more load or on numerous hosts/load engines only.
- Server mode for controlled distributed tests.
- It generate lots of load or be used with just a single user/thread if you are just interested in testing for specific responses or content.

3.13.1 JMeter Features

Some of the most important features of JMeter are listed below :

- **Open source application :** JMeter is a free open source application which facilitates users or developers to use the source code for development of other applications.
- **User-friendly GUI :** JMeter comes with simple and interactive GUI.
- **Support various testing approach :** JMeter supports various testing approach like Load Testing, Distributed Testing, and Functional Testing, etc.
- **Platform independent :** JMeter is written and developed using java, so it can run on any environment / workstation that accepts a Java virtual machine, for example - Windows, Linux, Mac, etc.
- **Support various server types :** JMeter is highly extensible and capable to load the performance test in different server types :
  - Web : HTTP, HTTPS, SOAP
  - Database : JDBC, LDAP, JMS
  - Mail : POP3.
- **Support multi-protocol :** JMeter supports protocols such as HTTP, JDBC, LDAP, SOAP, JMS, and FTP.
- **Simulation :** JMeter can simulate multiple users by using virtual users or unique users in order to generate heavy load against web application under test.
- **Framework :** JMeter is a multi-threading framework which allows concurrent and simultaneous sampling of different functions by many or separate thread groups.

- **Remote distributed testing :** JMeter has Master-Slave concept for distributed testing where master will distribute tests among all slaves and slaves will execute scripts against your server.
- **Test result visualization :** Test results can be viewed in different formats like graph, table, tree, and report etc.

3.13.2 Working of JMeter

- JMeter sends requests to a target server by simulating a group of users.
- Subsequently, data is collected to calculate statistics and display performance metrics of the target server through various formats.
- Fig. 3.13.1 shows workflow diagram of JMeter.

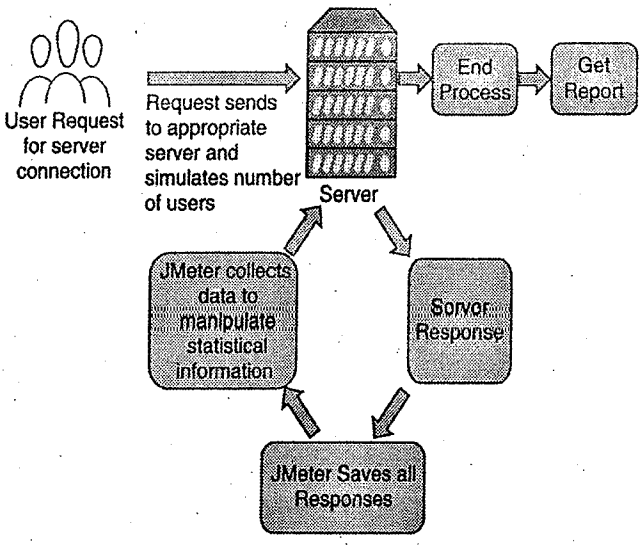


Fig. 3.13.1 : JMeter workflow diagram

3.13.3 Install JMeter

System Requirements

JMeter has the following system requirements :

- **Java Virtual Machine Installed :** JMeter is a Java-based application, It requires the Java Runtime to run,
- **CPU :** Mmulticore cpus with 4 or more cores. JMeter is heavily multithreaded and benefits from more CPU cores,
- **Memory : 16GB RAM** is a good value. It lets you simulate about 1000 concurrent users and leaving enough room for the Operating System,
- **Disk :** while JMeter not relies much on disk, having an SSD is a plus.
- **Network : 1Gbps LAN** recommended. JMeter simulates a huge number of concurrent users which are network bandwidth hungry.

Installation Steps

- Use latest JAVA version and latest jre.
- Set JAVA environment - Set the **JAVA_HOME** environment variable to point to the base directory location, where Java is installed on your machine and Java compiler location to System Path.
- Download JMeter from http://jmeter.apache.org/download_jmeter.cgi
- Run JMeter - go to the bin directory. In this case, it is /home/manisha/apache-jmeter-2.9/bin, then click on the JMeter.bat in windows, jmeter.sh in Linux and jmeter.sh in Mac.

3.13.4 JMeter - Testplan

- A test plan consists of test elements such as thread groups, logic controllers, sample-generating controllers, listeners, timers, assertions, and configuration elements.
- There should be at least one thread group in every test plan. We can add or remove elements as per our requirement.

Let's start building a Test plan by following these simple steps :

Step 1 : Launch the JMeter window

- Go to your JMeter bin folder and double click on the ApacheJMeter.jar file to launch JMeter interface.
- The default JMeter interface contains a Test Plan node where the real test plan is kept.
- The Test Plan node contains Name of the test plan and user defined variables.
- User defined variables provides flexibility when you have to repeat any value in several parts of the test plan.

Step 2 : Add/Remove test plan elements.

- Once you have created a test plan for JMeter, the next step is to add and remove elements to JMeter test plan.
- Select the test plan node and right click on the selected item.
- Mouse hover on "Add" option, then elements list will be displayed.
- Mouse hover on desired list element, and select the desired option by clicking.
- To remove an element, select the desired element.
- Right click on the element and choose the "Remove" option.

Step 3 : Load and save test plan elements.

- To load elements to JMeter test plan tree, select and right click on any Tree Element on which you want to add the loaded element.
- Select "Merge" option.
- Choose the .jmxfile where you save the elements.
- Elements will be merged into the JMeter test plan tree.
- To save tree elements, right click on the element.
- Choose "Save Selection As" option.
- Save file on desired location

Step 4 : Configuring the tree elements.

- Elements in the test plan can be configured by using controls present on JMeters right hand side frame.
- These controls allow you to configure the behaviour of the selected element. For example, a thread group can be configured by -
  - Its name.
  - Number of threads
  - Ramp-up time
  - Loop count

Step 5 : Save JMeter test plan.

- Till now we are done with creating a test plan, adding an element and configuring a Tree.
- Now, you can save the entire test plan by choosing the "Save" or "Save Test Plan As" from file menu.

Step 6 : Run JMeter test plan.

- You can run the test plan by clicking on the Start (Control + r) from the Run menu item or you can simply click the green play button.
- When the test plan starts running, the JMeter interface shows a green circle at the right-hand end of the section just under the menubar.
- The numbers to the left of the green circle represents : Number of active threads / Total number of threads

Step 7 : Stop JMeter test plan.

- You can stop the test plan by using Stop (Control + '.') - It stops the threads immediately if possible.
- You can also use Shutdown (Control + ',') - It requests the threads to stop at the end of any on-going task.

Step 8 : Check JMeter test plan execution logs.

- JMeter stores the test run details, warnings and errors to jmeter.log file.
- You can access JMeter logs by clicking on the exclamation sign present at the right-hand side of the section just under the menubar.

3.13.5 JMeter Test Plan Elements

- A Test Plan comprises of at least one Thread Group.
- Within each Thread Group, we may place a combination of one or more of other elements - Sampler, Logic Controller, Configuration Element, Listener, and Timer.
- Each Sampler can be preceded by one or more Pre-processor element, followed by Post-processor element, and/or Assertion element.

☞ **Following are the elements of JMeter Test Plan**

- Test Plan
- Thread Group
- Controllers
- Listeners
- Timers

- Configuration Elements
- Pre-Processor Elements
- Post-Processor Elements

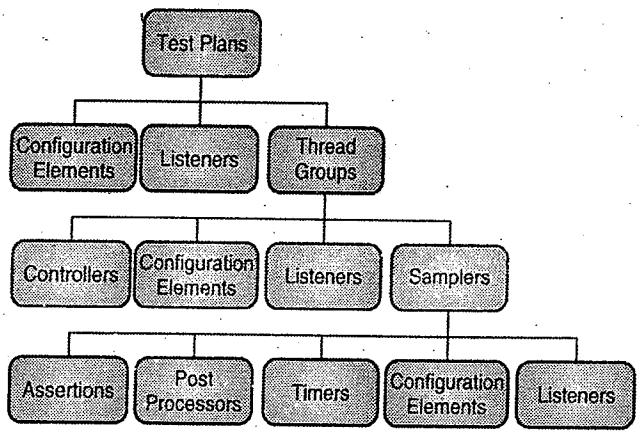


Fig. 3.13.2

1. Test Plan

- A test plan can be visualized as your JMeter script for running tests.
- A test plan consists of test elements such as thread groups, logic controllers, sample-generating controllers, listeners, timers, assertions, and configuration elements.
- A test plan consists of all steps which execute the script. Everything which is included in a test plan is executed in a sequence which is top to bottom or as per the defined sequence in the test plan.
- The test plan should be saved before running the entire test plan.
- JMeter files or test plans are saved in form of .JMX extension files. JMX is an open test based format, it enables the test plan to be launched in a text editor.
- You can also save parts of test plan as the different selection. For example, if you want to save HTTP request sampler with listener, you can save it as test fragment so that it can be used in other test scenarios as well.

2. Thread Group

- As the name implies, thread group represents the group of threads JMeter will use during the test. Thread group elements are the beginning points of any test plan.
- The controls provided by a thread group allow you to :
  - Set the number of threads.
  - Set the ramp-up period.
  - Set the number of times to execute the test.
- The steps to add a thread group in your test plan had been explained earlier in the Add/Remove test elements portion.

☞ **The Thread Group Control Panel includes**

- Its name.
- Number of threads means the number of users you are testing.
- Ramp-up time.
- Loop count mean how many times the test should be looped.
- Scheduler checkbox

3. Controllers

Controllers can be divided into two broad categories :

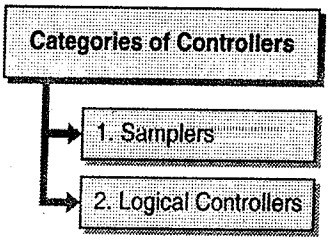


Fig. 3.13.3

➜ **1. Samplers**

- Samplers are the components which allow JMeter to send specific types of requests to a server. It simulates a user's request for a page to the target server.
- Samplers are a must to add component to a test plan as only it can let JMeter know what type of request need to go to a server. Requests could be HTTP, HTTP(s), FTP, TCP, SMTP, SOAP etc.
- Given below is the list of requests serviced by JMeter samplers :
  - FTP Request
  - HTTP Request (can be used for SOAP or REST Web service also)
  - JDBC Request
  - Java object request

- JMS request
- JUnit Test request
- LDAP Request
- Mail request
- OS Process request
- TCP request

2. Logical Controllers

- Logic Controllers help you to control the flow the order of processing of samplers in a thread.
- It can also change the order of requests coming from their child elements.

Following is the list of all Logic Controllers in JMeter

- Runtime Controller
- IF Controller
- Transaction Controller
- Recording Controller
- Simple Controller
- While Controller
- Switch Controller
- For Each Controller
- Module Controller
- Include Controller
- Loop Controller
- Once Only Controller
- Interleave Controller
- Random Controller
- Random Order Controller
- Throughput Controller

4. Listeners

- Performance testing is all about analysing server responses in various forms and then presenting the same to the client.
- Listeners provide pictorial representation of data gathered by JMeter about those test cases as a sampler component of JMeter is executed.
- It facilitates the user to view samplers result in the form of tables, graphs, trees or simple text in some log file.
- Listeners can be adjusted anywhere in the test, including directly under the test plan.
- There are around 15 listeners provided by JMeter but mostly used ones are table, tree, and Graph.

Following is the list of all Listeners in JMeter

- Graph Results
- Spline Visualizer
- Assertion Results
- Simple Data Writer
- Monitor Results
- Distribution Graph (alpha)
- Aggregate Graph
- Mailer Visualizer
- BeanShell Listener
- Summary Report
- Sample Result Save Configuration
- Graph Full Results
- View Results Tree
- Aggregate Report
- View Results in Table

5. Timers

- When you perform any operation on a website or app, they naturally have pauses and delays. These can be simulated with Timers.
- JMeter sends requests without applying any delay between each sampler/request.
- If you perform load/stress testing on your server without any delay, it will be overloaded. This not exactly what you want.
- You can add a timer element which will permit you to define a period to wait between each request.
- Given below is the list of all the Timer elements provided by JMeter :
  - Synchronizing Timer

- JSR223 Time
- BeanShell Time
- Gaussian Random Timer
- Uniform Random Timer
- Constant Throughput Timer
- BSF Time
- Poisson Random Time

6. Configuration Elements

- Working of configuration elements is quite similar to those of samplers. However, it does not send requests but it allows you to modify the requests made by the samplers.
- It is a simple element where you can collects the corporate configuration values of all samplers like webserver's hostname or database url etc.
- A configuration element is accessible from only inside the branch where you place the element.
- Given below is the list of some of the most commonly used configuration elements provided by Jmeter :
  - Java Request Defaults
  - LDAP Request Defaults
  - LDAP Extended Request Defaults
  - Keystore Configuration
  - JDBC Connection Configuration
  - Login Config Element
  - CSV Data Set Config
  - FTP Request Defaults
  - TCP Sampler Config
  - User Defined Variables
  - HTTP Authorization Manager
  - HTTP Cache Manager
  - HTTP Cookie Manager
  - HTTP Proxy Server
  - HTTP Request Defaults
  - HTTP Header Manager
  - Simple Config Element
  - Random Variable

7. Pre-processor Elements

- A Pre-Processor element is executed just before the request made by the sampler.
- If a Pre-processor is attached to a sampler element then it will execute just prior to that sampler element running.
- A Pre-processor element is used to modify the settings of a sample request just before it runs, or to update variables that are not extracted from response text.
- Following is a list of all Pre-processor elements provided by JMeter :
  - JDBC Pre-processor
  - JSR223 Pre-processor
  - RegEx User Parameters
  - BeanShell Pre-processor
  - BSF Pre-processor
  - HTML Link Parser
  - HTTP URL Re-writing Modifier
  - HTTP User Parameter Modifier
  - User Parameters

8. Post-processor Elements

- A Post-processor element is executed after a sampler request has been made. If a Post-Processor is attached to a Sampler element then it will execute just after that sampler element runs.
- A Post-processor is most often used to process the response data, for example, to extract a particular value for future purpose.
- Given below is the list of all Post-processor elements provided by JMeter :
  - CSS/JQuery Extractor
  - BeanShell Post-processor
  - JSR223 Post-processor
  - JDBC Post-processor
  - Debug Post-processor
  - Regular Expression Extractor
  - XPath Extractor

- Result Status Action Handler
- BSF Post-processor

JMeter Functions

JMeter functions can be referred as special values that can populate fields of any Sampler or other element in a test tree.

Syntax of a function in JMeter

${__functionName(var1,var2,var3)} ,

- Here "__functionName" matches the name of a function and Parentheses surround the parameters sent to the function.
- If a function parameter contains a comma then be sure to escape this with "\", otherwise JMeter will treat it as a parameter delimiter.

Review Questions

Q. Write a short note on : Test execution and reporting. **(Refer Section 3.1)**

Q. Explain design and architecture for automation. **(Refer Section 3.1)**

Q. List out challenges of automation testing. **(Refer Section 3.1)**

Q. Compare terms manual testing and automated testing. **(Refer Section 3.1)**

Q. Discuss "Generation of Automation". **(Refer Section 3.1)**

Q. Illustrate scope of automation in detail. **(Refer Section 3.1)**

Q. Explain automation framework in brief. **(Refer Section 3.1)**

Q. State various requirements for a Test Tool/ Framework. **(Refer Section 3.1)**

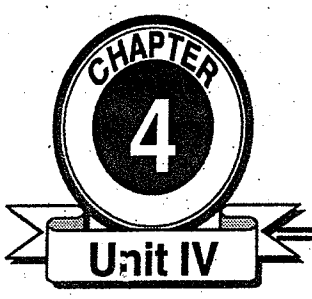
Q. Describe process model for automation testing? Explain W-model. **(Refer Section 3.7)**

Q. What are the different criteria's for selecting a testing tool? (Refer Section 3.8.2)

Q. Discuss automation in extreme programming. **(Refer Section 3.9)**

Q. List out challenges of automation testing. **(Refer Section 3.10)**

□□□



Selenium Tool

Syllabus Topics

Introducing Selenium, Brief History of the Selenium Project, Selenium's Tool Suite, Selenium-IDE, Selenium RC, Selenium Webdriver, Selenium Grid, Test Design Considerations

Syllabus Topic : Introducing Selenium

4.1 Introduction to Selenium

- In 2004, Jason R. Huggins and team is invented testing tool for web application and named as JavaScript Functional Tester.
- Selenium is a free open source web Automation tools or suit which is used to perform testing on Web Applications.
- It is similar to QTP but it only focuses on web-based applications.
- Selenium has a robust set of tools which supports rapid testing of test automation for web-based applications.
- It provides a rich set of testing functions specifically geared to the needs of testing of a web application.
- These operations are highly flexible, allowing many options for locating UI elements and comparing expected test results against actual application behavior.
- One of Selenium's key feature is the support for executing one's tests on multiple browser platforms.
- It is a single tool, but it is a suite of software to catering need of IT industry.

4.1.1 Features of Selenium

- Selenium is a Functional Automation tool for Web applications.
- Selenium is an open source tool.
- Selenium supports the languages like HTML, Java, PHP, Perl, Python, Ruby and C#.
- It supports the browsers like IE, Mozilla Firefox, Safari, Google Chrome and Opera.
- Used for functional regression testing.
- It supports the operating systems like Windows, Linux and Mac.
- It is very flexible when compared to QTP and other functional tools, because it supports multiple languages.
- Executing one's tests on multiple browser platforms.

Syllabus Topic : Brief History of the Selenium Project

4.1.2 Brief History of the Selenium Project

- In 2004 when Jason Huggins and his team was testing an internal application at ThoughtWorks. He is a engineer at ThoughtWorks.
- He was testing a web application that required frequent testing and it is very time consuming.
- How to utilize his time better way than manually stepping through the same tests with every change he made.
- Having realized that the repetitious Manual Testing of their application was becoming more and more inefficient.
- He created a **JavaScript** program that would automatically control the browser's actions. He named this program as the **"JavaScriptTestRunner"** and also allowing him to automatically rerun tests against multiple browsers.

- Selenium Core is born whose functionality underlies the Selenium RC (Remote Control) and Selenium IDE tools
- The Limitation of having a JavaScript based automation engine and browser security restricted Selenium to specific functionality
- Google, who has been a long time user of Selenium, had a developer named Simon Stewart who developed WebDriver. This tool circumvented Selenium's JavaScript sandbox to allow it to communicate with the Browser and Operating System directly using native methods
- In 2008, Selenium and WebDriver merged technologies and intellectual intelligence to provide the best possible test automation framework
- The merging of the two tools provided a common set of features for all users and brought some of the brightest minds in test automation under one roof.

Syllabus Topic : Selenium's Tool Suite

4.1.3 Selenium's Tool Suite

- **Selenium suit has four main components**
  - Selenium Integrated Development Environment (IDE)
  - Selenium Remote Control (RC)
  - WebDriver
  - Selenium Grid

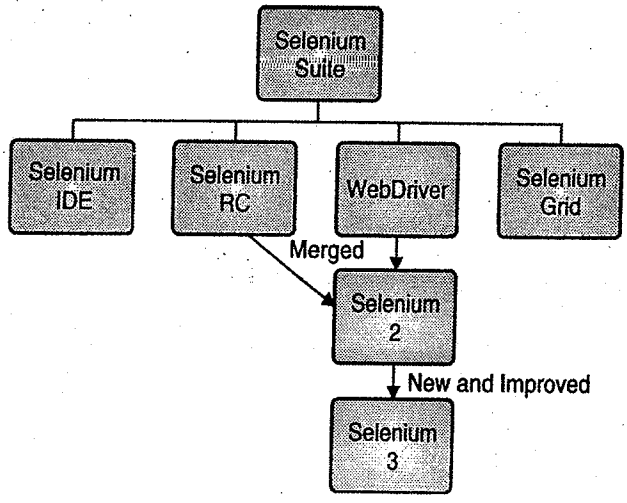


Fig. 4.1.1 : Selenium Tool Suite

Syllabus Topic : Selenium-IDE

4.2 Selenium-IDE

- Selenium IDE is an integrated development environment for Selenium tests. It is previously known as **Selenium Recorder.**
- It is implemented as a Firefox and Chrome extension, and allows you to record, edit, and replay the test in firefox and chrome. It allows easier development of tests.
- Selenium IDE allows you to save tests as HTML, Java, Ruby scripts, or any other format
- It allows you to automatically add assertions to all the pages.
- Allows you to add selenese commands as and when required and can be used by developers with little to no programming experience to write simple tests quickly and gain familiarity with the Selenese commands.
- Developed tests can be run against other browsers, using a simple command-line interface that invokes the Selenium RC server.
- Can export WebDriver or Remote Control scripts and these scripts should be in PageObject structure.
- Allows you the option to select a language for saving and displaying test cases.

4.2.1 Selenium IDE Features

- Record and playback.
- Intelligent field selection will use IDs, names, or XPath as needed.
- Auto complete for all common Selenium commands.
- Walk through test cases and test suites.
- Debug and set breakpoints.
- Save tests as HTML, Ruby scripts, or other formats.
- Support for Selenium user-extensions.js file.
- Option to automatically assert the title of every page.
- Rollup common commands.

4.2.2 Selenium IDE Installation

1. Using Firefox or Chrome, first, download the IDE from the SeleniumHQ downloads page when downloading from Firefox, you'll be presented with the Fig. 4.2.1.

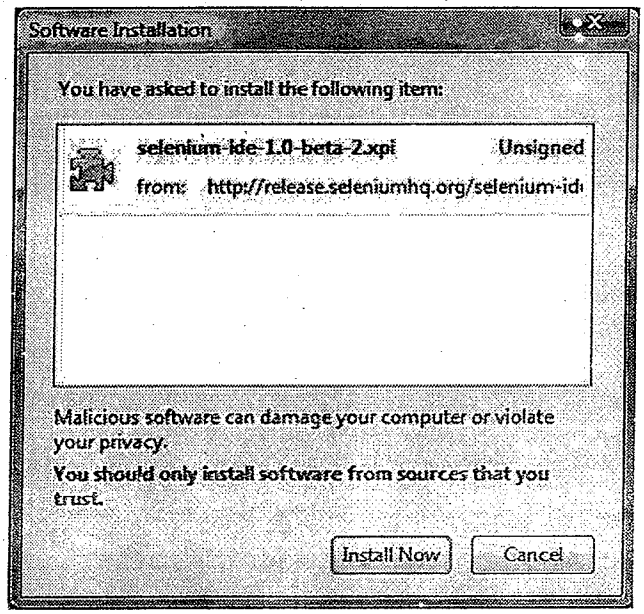


Fig. 4.2.1 : Selenium installation window

2. Select Install Now. The Firefox or Chrome Add-ons window pops up, first showing a progress bar, and when the download is complete, displays the Fig. 4.2.2.

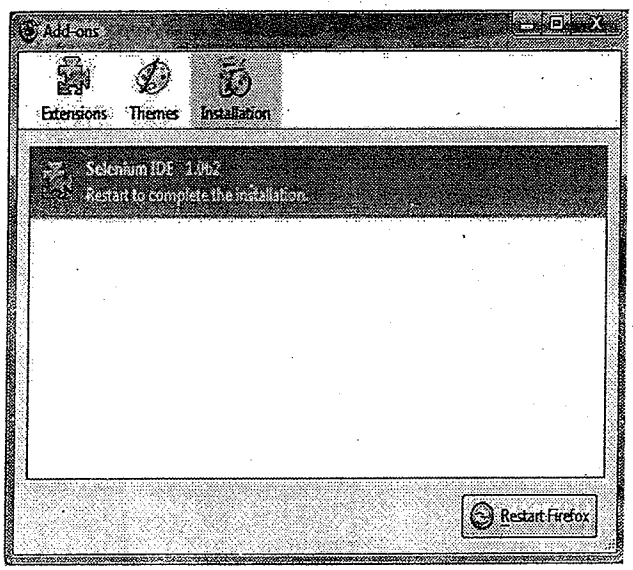


Fig. 4.2.2 : Selenium installed and asking to restart

3. Restart Firefox. After Firefox reboots you will find the Selenium-IDE listed under the Firefox or Chrome Tools menu.

4.2.3 Selenium IDE – UI

– Selenium different user interface shown in Fig. 4.2.3.

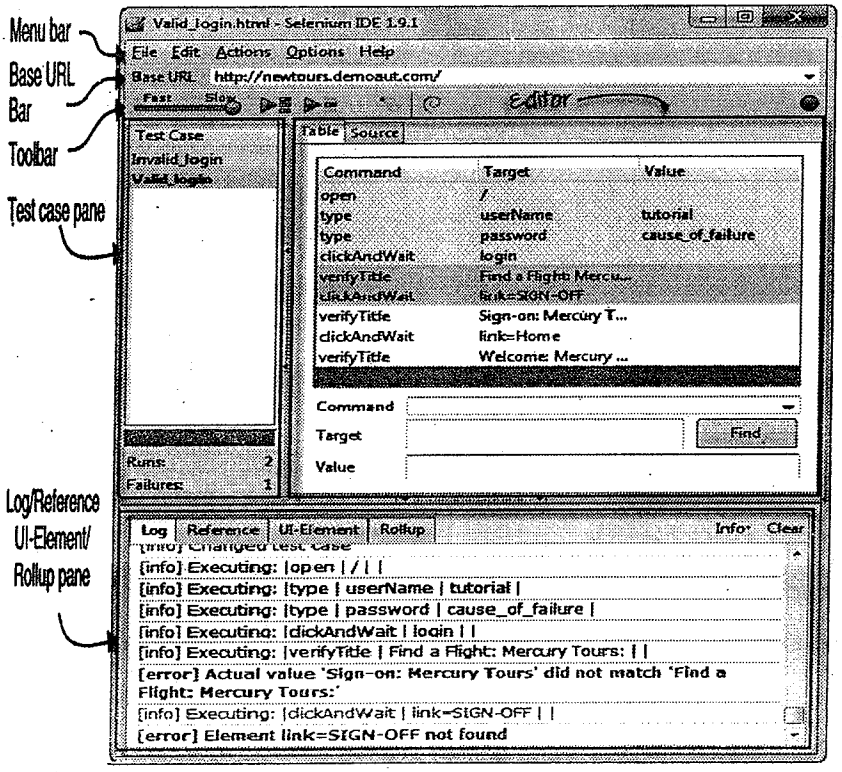


Fig. 4.2.3

Menu Bar

It is located at the **topmost portion** of the IDE. The most commonly used menus are the File, Edit, and Options menus.

File menu

- It contains options to create, open, save, and close tests.
- Tests are **saved in HTML format.**
- The most useful option is "Export" because **it allows you to turn your Selenium IDE test cases into file formats that can run on Selenium Remote Control and WebDriver.**
  - **"Export Test Case As..."** will export only the currently opened test case.
  - **"Export Test Suite As..."** will export all the test cases in the currently opened test suite shown in Fig. 4.2.4.

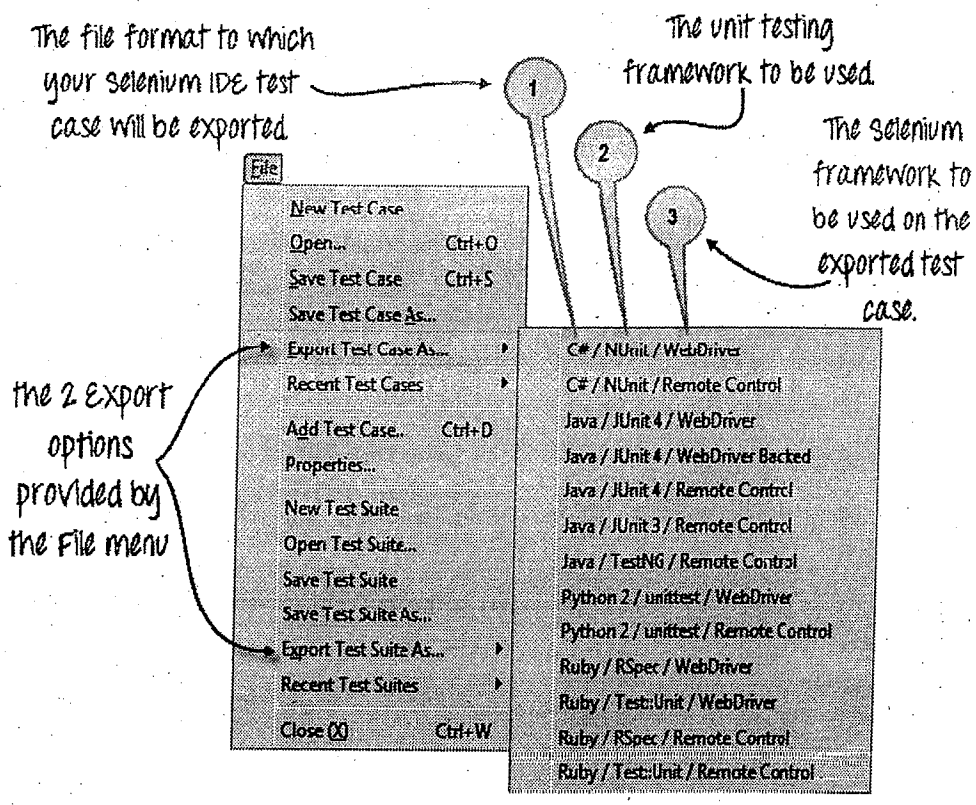


Fig. 4.2.4

Edit Menu

- It contains usual options like Undo, Redo, Cut, Copy, Paste, Delete, and Select All.
- The two most important options are the **"Insert New Command"** and **"Insert New Comment"**.

Options menu

- It provides the **interface for configuringvarious settings** of Selenium IDE.
- We shall concentrate on the **Options** and **Clipboard Format** options shown in Fig. 4.2.5.

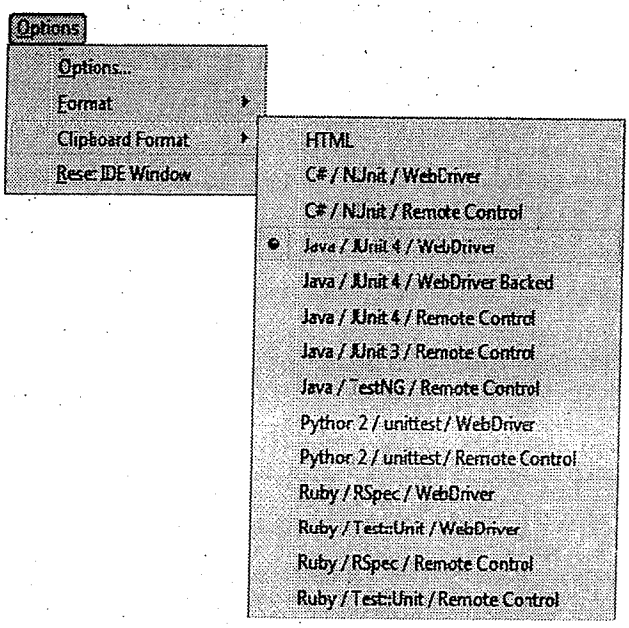


Fig. 4.2.5

Toolbar

Fast Slow	**Playback Speed :** This controls the speed of your Test Script Execution
	Record : This starts/ends your recording session. Each browser action is entered as a Selenese command in the Editor.
	Play entire test suite : This will sequentially play all the test cases listed in the Test Case Pane.
	Play current test case : This will play only the currently selected test case in the Test Case Pane.
	Pause/Resume : This will pause or resume your playback.
	Step : This button will allow you to step into each command in your test script.
	Apply rollup rules : This is an advanced functionality. It allows you to group Selenese commands together and execute them as a single action.

Test Case Pane

- In Selenium IDE, you can open **more than one test case at a time** shown in Fig. 4.2.6.
- **The test case pane shows you the list of currently opened test cases.**
- When you open a test suite, the test case pane will **automatically list all the test cases** contained in it.
- The test case written in **bold font** is the **currently selected** test case.
- After playback, **each test case is color-coded** to represent if it passed or failed.
  - Green color means "Passed."
  - Red color means "Failed."
- At the bottom portion is a summary of the number of test cases that were run and failed.

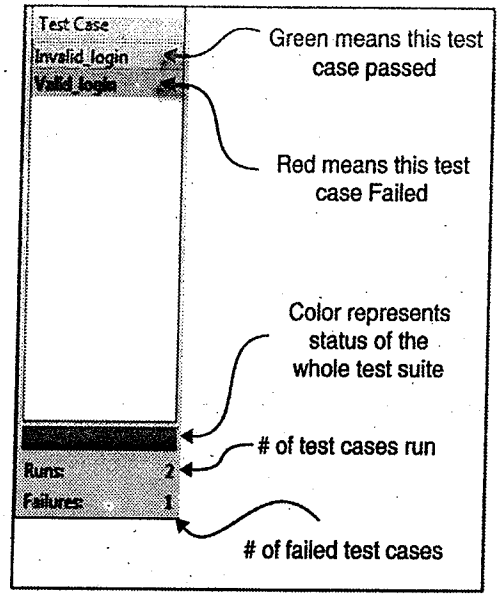


Fig. 4.2.6

Editor

The editor used as **the place where all the action happens.** It is available in two views: Table and Source shown in Fig. 4.2.7 .

Table View

- Most of the time, you will work on Selenium IDE using the Table View.
- This is where you create and modify Selenese commands.
- After playback, each step is color-coded.

Source View

- **It displays the steps in HTML (default) format.**
- **It also allows you to edit your script** just like in the Table View.

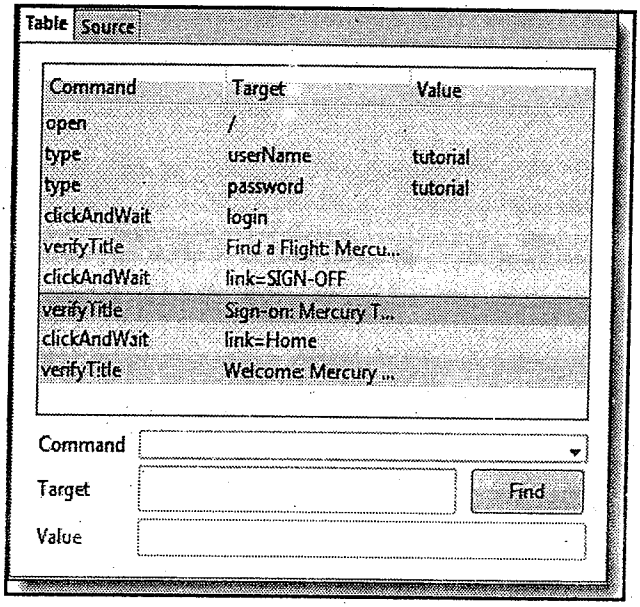


Fig. 4.2.7 : Editor Pane

Log Pane

- **The Log Pane displays runtime messages** during execution. It provides real-time updates as to what Selenium IDE is doing.

Logs are categorized into four types

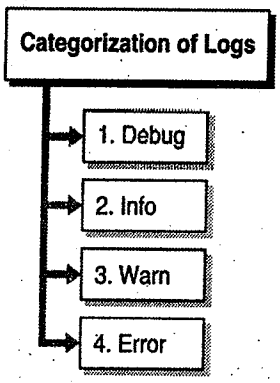


Fig. 4.2.8 : Categorization of logs

1. Debug

- By default, Debug messages are not displayed in the log panel. They show up only when you filter them. They provide technical information about what Selenium IDE is doing behind the scenes.
- It may display messages such as a specific module has done loading, a certain function is called, or an external JavaScript file was loaded as an extension.

2. Info

- It says which command Selenium IDE is currently executing.

3. Warn

- These are warning messages that are encountered in special situations.

4. Error

- These are error messages generated when Selenium IDE fails to execute a command, or if a condition specified by "verify" or "assert" command is not met.

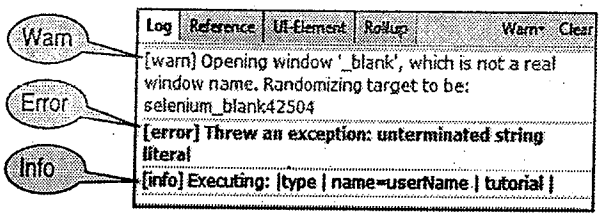


Fig. 4.2.9

4.2.4 Recoding a Selenium Test Case

- Open Firefox or Chrome that has the IDE installed
- Open the base URL of the application to record.
- Keep the application in a common base state.
- Go To Tools → Selenium IDE and the IDE will be opened.
- Now perform the operations on the application as you are testing the application.
- Once you are done with the recording click on the stop recording button and save the test case through the file menu. By default it will be saved as a selenese script in HTML format.

Test Case Running Options

- **Run a Test Case :** Click the Run button to run the currently displayed test case.
- **Run a Test Suite :** Click the Run All button to run all the test cases in the currently loaded test suite.
- **Stop and Start :** The Pause button can be used to stop the test case while it is running. The icon of this button then

changes to indicate the Resume button. To continue click Resume.

- **Stop in the Middle :** You can set a breakpoint in the test case to cause it to stop on a particular command. This is useful for debugging your test case. To set a breakpoint, select a command, right-click, and from the context menu select Toggle Breakpoint.

4.2.5 Selenium Commands – "Selenese"

- Selenium commands also called *selenese*, are the set of commands that run your tests.
- A sequence of these commands is a *test script*. Selenium provides a rich set of commands for fully testing your web-app in virtually any way you can imagine. The command set is often called *selenese*. These commands essentially create a testing language.
- A *command* tells Selenium what to do.
- Selenium commands come in three "flavors": **Actions, Accessors, and Assertions.**

1. **Actions** are commands that generally manipulate the state of the application. They do things like "click this link" and "select that option". If an Action fails, or has an error, the execution of the current test is stopped.
2. **Accessors** examine the state of the application and store the results in variables, e.g. "storeTitle". They are also used to automatically generate Assertions.
3. **Assertions** are like Accessors, but they verify that the state of the application conforms to what is expected. Examples include "make sure the page title is X" and "verify that this checkbox is checked".

All Selenium Assertions can be used in 2 modes: "assert" and "verify". For example, you can "assertText" and "verifyText". When an "assert" fails, the test is aborted. When a "verify" fails, the test will continue execution, logging the failure. This allows a single "assert" to ensure that the application is on the correct page, followed by a bunch of "verify" assertions to test form field values, labels, etc.

General Selenese Commands

- clicking a link - *click* or *clickAndWait* commands.
- entering values - *type* command.
- selecting options from a drop-down listbox - *select* command.
- clicking checkboxes or radio buttons - *click* command.

Store Commands

- Store commands are used to fetch the values from the application and store it in a variable. These variables can be used latter for validation purpose.
- The Store command can be used to retrieve the page title, text from the page and other attributes from the application.

Echo Command

- Echo command is used to print the value in to the selenium IDS log.
- When printing a variable use ${var}.
- There are some limitations for this methods this has to be used with caution.

Syllabus Topic : Selenium RC

4.3 Selenium RC

- **Selenium Remote Control (RC)** is a server, written in Java that accepts commands for the browser via HTTP.
- RC makes it possible to write automated tests for a web application in any programming language, which allows for better integration of Selenium in existing unit test frameworks.
- Selenium-RC (Remote Control) provides an API (Application Programming Interface) and library for each of its supported languages: HTML, Java, C#, Perl, PHP, Python, and Ruby.
- This ability to use Selenium-RC with a high-level programming language to develop test cases also allows the automated testing to be integrated with a project's automated build environment.
- It allows you to write automated web application UI tests in any programming language against any HTTP website using any mainstream JavaScript-enabled browser.
- The primary task for using Selenium RC is to convert your Selenese into a programming language
- It provides solution to cross browser testing.
- A server, written in Java and so available on all the platforms.

- Acts as a proxy for web requests from them.
- Client libraries for many popular languages.
- Bundles Selenium Core and automatically loads into the browser.

4.3.1 Components of Selenium RC

Selenium-RC components are shown in Fig. 4.3.2

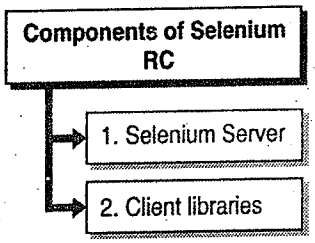


Fig. 4.3.1 : Components of Selenium RC

➜ **1. The Selenium Server**

- This launches and kills browsers, interprets and runs the Selenese commands passed from the test program, and acts as an HTTP proxy, intercepting and verifying HTTP messages passed between the browser and the AUT.

➜ **2. Client libraries**

- Which provide the interface between each programming language and the Selenium-RC Server.
- Fig. 4.3.1 shows the client libraries communicate with the Server passing each Selenium command for execution.
- Then the server passes the Selenium command to the browser using Selenium-Core JavaScript commands.
- The browser, using its JavaScript interpreter, executes the Selenium command.
- This runs the Selenese action or verification you specified in your test script.

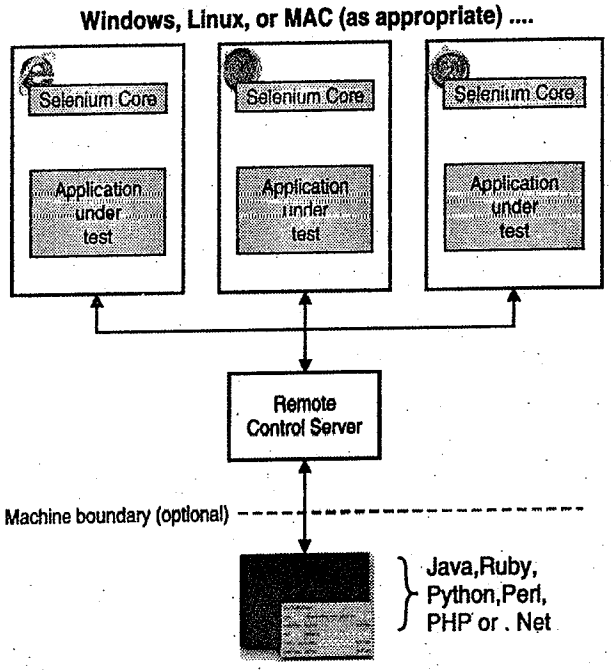


Fig. 4.3.2: Selenium RC Components

4.3.1(A) Selenium Server

- Selenium Server receives Selenium commands from your test program, interprets them, and reports back to your program the results of running those tests.
- The RC server bundles Selenium Core and automatically injects it into the browser. This occurs when your test program opens the browser using a client library API function.
- Selenium-Core is a JavaScript program, actually a set of JavaScript functions which interprets and executes Selenese commands using the browser's built-in JavaScript interpreter.
- The Server receives the Selenese commands from your test program using simple HTTP GET/POST requests. This

means you can use any programming language that can send HTTP requests to automate Selenium tests on the browser.

4.3.1(B) Client Libraries

- The client libraries provide the programming support that allows you to run Selenium commands from a program of your own design.
- There is a different client library for each supported language. A Selenium client library provides a programming interface (API), i.e., a set of functions, which run Selenium commands from your own program.
- Within each interface, there is a programming function that supports each Selenese command.
- The client library takes a Selenese command and passes it to the Selenium Server for processing a specific action or test against the application under test (AUT). The client library also receives the result of that command and passes it back to your program. Your program can receive the result and store it into a program variable and reporting it as a success or failure, or possibly taking corrective action if it was an unexpected error.
- So to create a test program, you simply write a program that runs a set of Selenium commands using a client library API. And, optionally, if you already have a Selenese test script created in the Selenium-IDE, you can generate the Selenium-RC code. The Selenium-IDE can translate using its Export menu Item, its Selenium commands into a client-driver's API function calls.

4.3.2 Installing Selenium RC

Software Required

- Download JDK 1.6 and above and Set path and variables.
- Download Selenium-remote-control 3.0 and above **http://seleniumhq.org/download/**
- Selenium RC is simply a jar file and to run it we need java installed. (JDK 1.6 and above is preferred).
- Once the Java is installed just unzip the selenium-remote-control-3.0 and above zip which was downloaded from the selenium site to a directory.
- Once user decided language to work with user need to:
  - Install Selenium RC Server
  - Language Specific Client Driver

Install Selenium RC Server

- Before starting any tests you must start the server.
- Go to the directory where Selenium RC's server is located and run the following from a command-line console.

  java -jar selenium-server-standalone-<version-number>.jar
- This can be simplified by creating a batch or shell executable file .bat on Windows and .sh on Linux.
- For the server to run you'll need Java installed and the PATH environment variable correctly configured to run it from the console. You can check that you have Java correctly installed by running the following on a console.

  java -version

Syllabus Topic : Selenium Webdriver

4.4 Selenium Webdriver

- Selenium driver is designed to address the limitations of Selenium RC. It is also called Selenium 2 and is the successor to Selenium RC.
- It is also designed to support dynamic web pages and control the browser by programming. Makes direct calls to the browser using each browser's native support for automation.
- WebDriver's goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.
- Selenium-WebDriver supports multiple browsers in multiple platforms such as Google Chrome 12.0.712.0+, Internet Explorer 6+, Firefox 3.0+, Opera 11.5+, Android – 2.3+ for phones and tablets, iOS 3+ for phones, iOS 3.2+ for tablets.
- WebDriver interacts directly with the browser without any intermediary, unlike Selenium RC that depends on a server.
- It is used in the following context –
  - Multi-browser testing including improved functionality for browsers which is not well-supported by Selenium RC (Selenium 1.0).

- Handling multiple frames, multiple browser windows, popups, and alerts.
- Complex page navigation.
- Advanced user navigation such as drag-and-drop.
- AJAX-based UI elements.

4.4.1 Advantages of Selenium WebDriver

- Scripts written to perform browser actions to simulate web user.
- Tests against various browsers and devices.
- Flexible to handle frequent code changes.
- Watch scripts run against live browser.
- Scalable with Selenium Grid.

4.4.2 Disadvantages of Selenium WebDriver

- Simulates user actions but does not support scrolling.
- Must hack shortcomings with Javascript.
- WebDriver tends to be out of date with frequent browser updates.

4.4.3 Difference between WebDriver vs Selenium RC

Sr. No.	Selenium WebDriver	Selenium RC
1.	Architecture of Selenium WebDriver is simple.	Architecture of Selenium RC is complicated.
2.	WebDriver interacts directly with the browser and uses the browser's engine to control it	Selenium server acts as a middleman between the browser and Selenese commands.
3.	WebDriver is faster, as it interacts directly with the browser.	Selenium RC script execution is slower, since it uses a Javascript to interact with RC.
4.	Object-oriented API	Dictionary-based API
5.	Only supported Java	Wide-range of languages
6.	Bind natively to the browser	Selenium Core (JavaScript)
7.	Side-stepping the browser's security model	Playing with browser's security model
8.	It can test mobile applications.	It cannot test mobile applications.

4.4.4 Difference between Selenium IDE, Selenium RC and WebDriver

Selenium IDE	Selenium RC	Webdriver
Works only on mozilla	Works on almost all browsers.Does not work on latest version of firefox/IE	Works on latest versions of almost all browsers - Firefox, IE(6,7,8), OPera, Chrome
Record and run tool	No Record and run	No Record and run
No server required to start	Server is required to start	No server required to start
Core engine is Javascript based	Core engine is Javascript based	Interacts natively with browser application
Very simple to use.	Its a simple and small API	Complex and a bit large API as compared to RC
Not at all object oriented	Less Object orinted API	Purely Object oriented API
Cannot move mouse with it	Cannot move mouse with it	Can move mouse cursor
Full xpaths have to be appended with 'xapth=\\' syntax	Full xpaths have to be appended with 'xapth=\\' syntax	No need to append 'xpath=\\'

4.4.5 Installation of Selenium WebDriver

Prerequisites

- Mozilla Firefox
- Active Internet Connection

Installation Steps

1. **Download and Install JAVA**
2. Download Eclipse IDE for Java Developers and Install
3. **Download and Install FireBug**

   (https://addons.mozilla.org/en-US/firefox/addon/firebug/)
4. **Download and Install FirePath**

   (https://addons.mozilla.org/en-US/firefox/addon/firepath/)
5. Download and Install Selenium WebDriver

4.4.6 Selenium WebDriver Architecture

- Selenium is a browser automation tool which interacts with browser and automate end to end tests of a web application.
- Selenium Webdriver API helps in communication between languages and browsers.
- Selenium supports many programming languages such as Java, C#, Python etc., and also it supports multiple browsers such as Google Chrome, Firefox, Internet Explorer etc.
- Fig. 4.4.1 shows architecture of selenium WebDriver.
- Selenium WebDriver architecture has four components :
  1. Selenium Client Library
  2. JSON Wire Protocol over HTTP
  3. Browser Drivers
  4. Browsers

1. Selenium Client Libraries/Language Bindings

Selenium supports multiple libraries such as Java, Ruby, Python, etc., Selenium Developers have developed language bindings to allow Selenium to support multiple languages.

2. JSON WIRE PROTOCOL Over HTTP Client

JSON stands for JavaScript Object Notation. It is used to transfer data between a server and a client on the web. JSON Wire Protocol is a REST API that transfers the information between HTTP server. Each BrowserDriver (such as FirefoxDriver, ChromeDriver etc.,) has its own HTTP server.

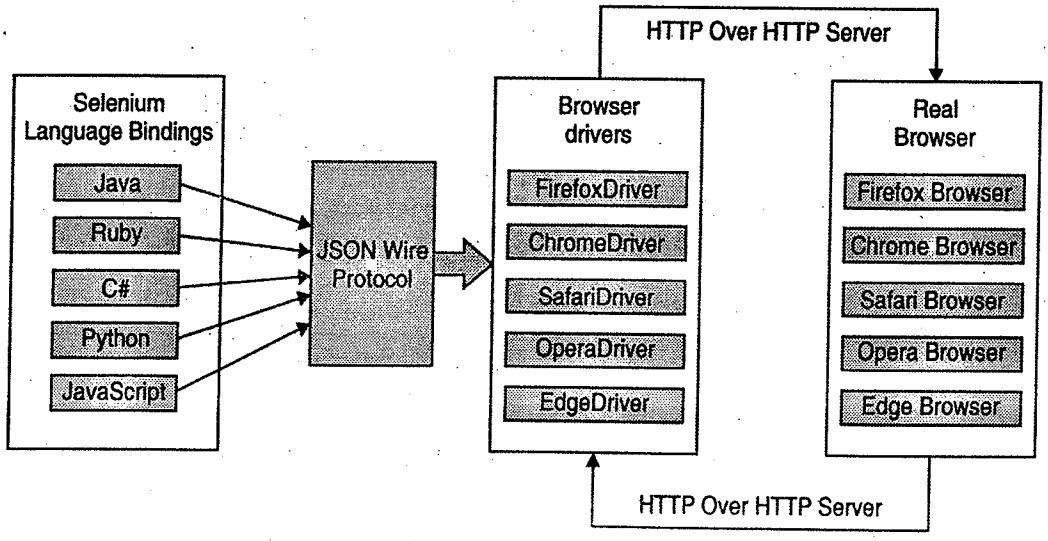


Fig. 4.4.1 : Selenium WebDriver Architecture

3. Browser Drivers

Each browser contains separate browser driver. Browser drivers communicate with respective browser without revealing the internal logic of browser's functionality. When a browser driver is received any command then that command will be executed on the respective browser and the response will go back in the form of HTTP response.

4. Browsers

Selenium supports multipe browsers such as Firefox, Chrome, IE, Safari etc.

Syllabus Topic : Selenium Grid

4.5 Selenium Grid

- Selenium Grid is a tool used together with Selenium RC to run tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems shown in Fig. 4.5.1.

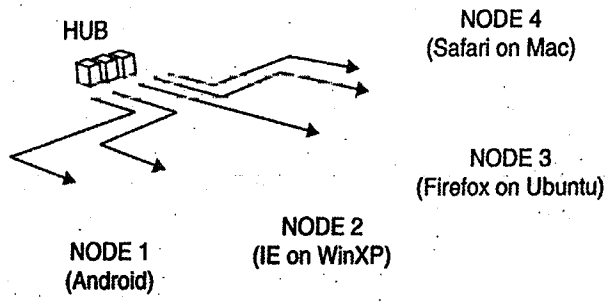


Fig. 4.5.1 : Selenium Grid for multiple node parallel

- Selenium Grid is capable of coordinating WebDriver tests/ RC tests which can run simultaneously on multiple web browsers or can be initiated on different operating systems or even hosted on different machines.

☞ Why and When To Use Selenium Grid?

- When to run our tests against multiple browsers, the multiple versions of browsers and the browsers running on different operating system.
- It is also used to reduce the time taken by the test suite to complete a test pass by running tests in parallel.

- Selenium Grid has two versions:
  - Selenium Grid 1
  - Selenium Grid 2

☞ Difference between Selenium Grid 1 and 2

Sr. No.	Selenium Grid 1	Selenium Grid 2
1.	Requires Apache Ant to be installed	Apache Ant installation is not required
2.	Has its own remote control (This is different from RC server)	Bundled with Selenium Server jar file
3.	Supports only Selenium RC commands	Supports both Selenium RC and WebDriver scripts
4.	You can automate only one browser per remote control	You can automate up to 5 browsers per remote control

4.5.1 Selenium Grid Architecture

- Selenium Grid has two main components :
  - Hub
  - Node

☞ 1. Hub

- The hub is the central point where you load your tests into.
- There should only be one hub in a grid.
- Hub also acts as a server because of which it acts as a central point to control the network of Test machines.
- The hub is launched only on a single machine, say, a computer whose O.S is Windows 7 and whose browser is IE.
- The machine containing the hub is where the tests will be run, but you will see the browser being automated on the node.

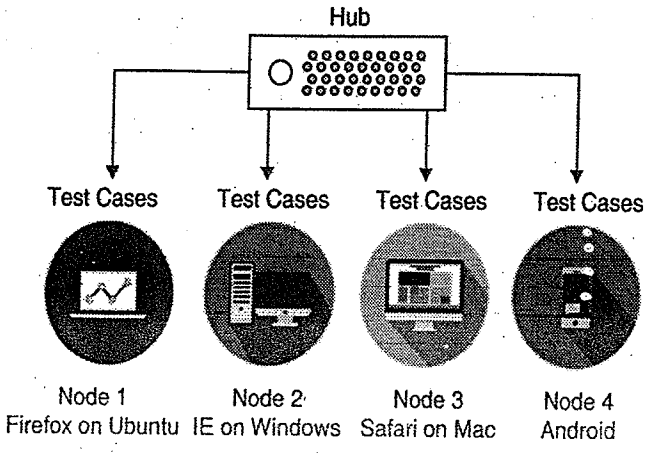


Fig. 4.5.2: Selenium Grid Architecture

2. Node

- Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
- There can be one or more nodes in a grid.
- Nodes can be launched on multiple machines with different platforms and browsers.
- The machines running the nodes need not be the same platform as that of the hub.

4.5.2 Installation of Selenium Grid

Selenium Grid installation steps are

Step 1 : Download Selenium Server jar file from Selenium's official website which is formerly known as Selenium RC Server and save it at any location on the local disk.
URL of selenium Grid :
http://www.seleniumhq.org/download/

Step 2 : Open the command prompt and navigate to a folder where the server is located. Run the server by using below command *java -jar selenium-server-standalone-2.41.0.jar -role hub*

- The hub will use the port 4444 by default. This port can be changed by passing the different port number in command prompt provided the port is open and has not been assigned a task.
- Status can be checked by using the web interface: *http://localhost:4444/grid/console*

Step 3 : Go to the other machine where you intend to setup Nodes. Open the command prompt and run the below line.

- java -jar selenium-server-standalone-2.41.0.jar -role node -hub
- http://localhost:4444/grid/register -port 5556
- Run the selenium server in other machines to start nodes.

Syllabus Topic : Test Design Considerations

4.6 Test Design Considerations

- Test design consideration describes most common types of automated tests. This provides useful information to new tester as well as experience also.
- This improves the maintenance and extensibility of your automation suite.

Testing Static Content

- Static content testing is a simple type of testing to test static, non-changing, UI element of web applications.
- It tests:
  - Does each page have its expected page title? This can be used to verify your test found an expected page after following a link.
  - Does the application's home page contain an image expected to be at the top of the page?

- Does each page of the website contain a footer area with links to the company contact page, privacy policy, and trademarks information?
- Does each page begin with heading text using the <h1> tag? And, does each page have the correct text within that header?

Function Tests

- Function test is used to test a specific function within application, requiring some type of user input, and returning some type of results.
- Often a function test will involve multiple pages with a form-based input page containing a collection of input fields, Submit and Cancel operations, and one or more response pages.
- User input can be via text-input fields, check boxes, drop-down lists, or any other browser-supported input.
- Function tests are often the most complex tests and the most important.
- Typical tests can be for login, registration to the site, user account operations, account settings changes, complex data retrieval operations, among others.
- Function tests typically mirror the user-scenarios used to specify the features and design or your application.

Review Questions

Q. What is Selenium? What are the different features of it ? **(Section 4.1)**

Q. What is Selenium? Draw a neat diagram to represent Test Suite. **(Section 4.3)**

Q. Describe Selenium RC in brief. **(Section 4.3)**

Q. Differentiate between WebDriver vs Selenium RC. **(Section 4.4.3)**

Q. Differentiate between Selenium IDE, Selenium RC and WebDriver. **(Section 4.4.4)**

Q. State various advantages and disadvantages of Selenium WebDriver. **(Sections 4.4.1 & 4.4.2)**

Q. Describe Selenium WebDriver Architecture with the help of neat diagram. **(Section 4.4.6)**

Q. Explain difference between Selenium Grid 1 and 2. **(Section 4.5)**

Q. What are the various Test Design Considerations? **(Section 4.6)**

Q. DescribeSelenium WebDriver in brief. **(Section 4.4)**

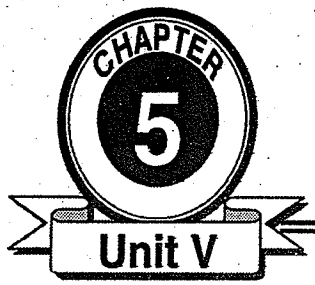
Q. Explain Selenium Grid Architecture with the help of neat diagram. **(Section 4.5.1)**

Q. Why should Selenium be selected as a test tool? **(Section 4.1.3)**

Q. What is Selenium? What are the different Selenium components? **(Sections 4.1 & 4.3.1)**

Q. When should you use Selenium Grid? **(Section 4.5)**

□□□



Quality Management

Syllabus Topics

Software Quality, Software Quality Dilemma, Achieving Software Quality, Software Quality Assurance. Elements of SQA, SQA Tasks, Goals, and Metrics, Formai Approaches to SQA, Statistical Software Quality Assurance, Six Sigma for Software Engineering, ISO 9000 Quality Standards, SQA Plan.

Syllabus Topic : Software Quality

5.1 Introduction to Quality Concepts

5.1.1 Software Quality

☞ **Software**

According to the IEEE Standard 729-1983, software is "computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer System."

☞ **Software Quality Definition**

- It is the ability of the software to comply with defined requirements.
- IEEE definition of Software quality is :
  1. The degree to which a system, component, or process meets specified requirements.
  2. The degree to which a system, component, or process meets customer or user needs or expectations.
- The ISO had defined quality as, the totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs.
- Following are the definitions given by quality specialists as follows :
  - Conformance to user requirements - Phil Crosby
  - Achieving excellent levels of fitness for use, conformance to requirements, reliability and maintainability - Watts Humphrey
  - Being on time, within budget and meeting user needs - James Martin
  - High levels of user satisfaction and low level defects, often associated with low complexity - Tom McCabe.
- The most general software quality standard are ISO 9001 and British Standard Institute BS5750.

☞ **The relevant standards for the industry**

- In general way, software quality is defined as, An effective software process applied in a systematic way that creates a useful product that provides measurable value to organization who produce it and end user who use it.
- An effective process forms infrastructure contains umbrella activities such as technical review, change management, process assessment, process improvement, problem analysis, design etc.

- A useful product delivers functionality, feature, content etc. Desired by end customer.
- High quality product is beneficial for both developing organization and end user. High quality ensures less defects fixing, low maintenance effort, minimum customer support. This causes them to focus on creating more applications.

5.1.1(A) Garvin's Quality Dimensions

- David Garvin had suggested eight dimensions of quality applied when software quality is considered.
- According to Garvin, quality should be considered by taking a multidimensional view point where assessment of conformance is done at beginning and ends with an aesthetic or transcendental view.

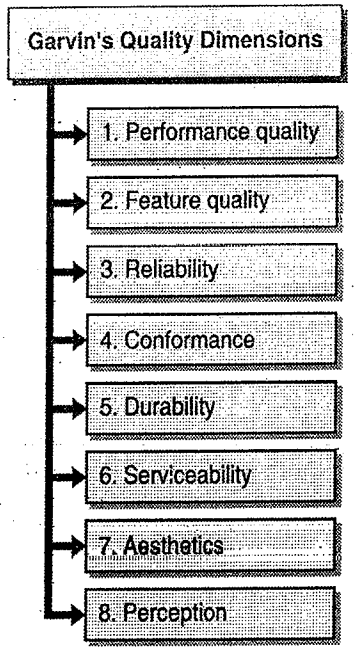


Fig. 5.1.1 : Garvin's Quality Dimensions

➜ **1. Performance quality**

Does software delivering all functions, features and contents, as per requirement specified and that provides value to the end user?

➜ **2. Feature quality**

Does the software has features that will surprise and delight first-time end customer?

➜ **3. Reliability**

Does the software deliver all features and the capability without failure? Is it available when needed? Does it deliver error-free functionality?

➜ **4. Conformance**

Does the software conform to application related local and external software standards? Does it conform to design and coding conventions?

➜ **5. Durability**

Can the software be changed/maintained or debugged/corrected without introducing unintended side effects? Will the changes low down the error rate and reliability with time?

➜ **6. Serviceability**

Can the software be changed/maintained or debugged/corrected within acceptable short time span? Can support staff collect all required information to make changes or correct defects?

➜ **7. Aesthetics**

Software with characteristics like, elegance, a unique flow, an obvious presence which are hard to measure or quantify.

➜ **8. Perception**

In some situations, past experiences will influence the perception of quality. e.g. A software product by a vendor who has produced poor quality in the past, will make a perception about the current software product quality in a negative way and vice versa.

- Garvin's quality dimensions provides soft look of software quality and does not focus on measurable attributes. So there must be a set of hard quality factors that can be measurable in a direct and indirect way.

5.1.1(B) McCall's Quality Factors

It was developed in 1976-77 and one of the oldest software quality models. It is also called as General Electric's Model. It was mainly developed for US military to bridge the gap between users and developers.

Model is comprised of -

- A volume of 55 quality characteristics ("factors") having an important influence on quality.
- The quality factors were compressed into 11 main factors in order to simplify the model.
- The quality of software products was defined according to 03 major perspectives as - product revision (ability to undergo changes), product transition (adaptability to new environments), product operations (its operation characteristics).
- 11 factors which describe the external view of the software (known as user view),
- 23 quality criteria which describe the internal view of the software (known as developer view) metrics that are used to provide a scale and method for measurement.

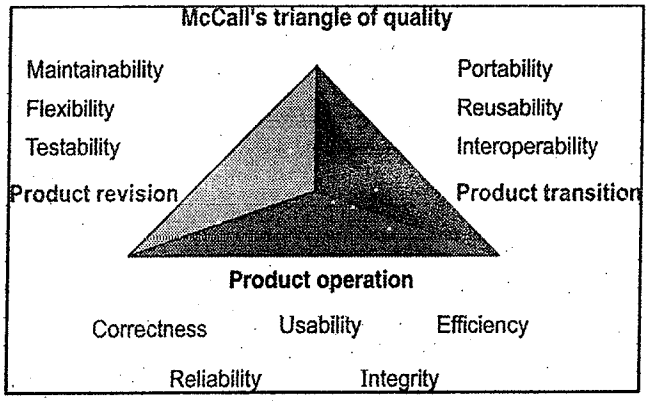


Fig. 5.1.2 : McCall's Product Quality Model

The three major perspectives of model are as follows :

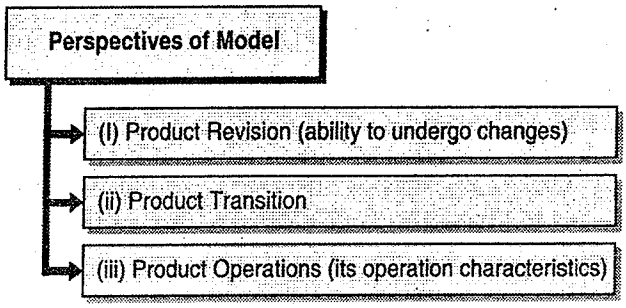


Fig. 5.1.3 : Perspectives of Model

(i) Product Revision (ability to undergo changes)

It encompasses the revision perspective quality factors. These factors changes or enhances the ability to change the software product in the future as per the needs and requirements of the user.

Table 5.1.1 : Product revision attributes

1.	Maintainability	Ability to find and fix the defects.
2.	Flexibility	Ability to make changes in the software product as per business demands.
3.	Testability	Ability to validate the software product requirements.

(ii) Product Transition

It enables the software to adapt itself in new environments.

Table 5.1.2 : Product transition attributes

1.	Portability	This is the ability to transfer a software from one environment to another environment.
2.	Re-usability	The software components can be used in different contexts.
3.	Interoperability	The ease or the comfort zone in which all the components of the software works together.

(iii) Product Operations (its operation characteristics)

The software can run successfully in the market if it according to the specifications of the user and also it should run smoothly without any defects. The product operation perspective focuses on the software fulfills its specifications.

Table 5.1.3 : Product operations attributes

1.	Correctness	The functionality should match the specification.
2.	Reliability	The extent to which the system fails.
3.	Efficiency	It enhances the usage of system resource.
4.	Usability	The software should be easy to use. Difficult software is tedious to work upon and difficulty irks the user.
5.	Integrity	The protection of program from unauthorized access.

Major contribution of this model is that, it provides the relationship between the quality characteristics and metrics. However, it is not considering the functionality of software products.

5.1.1(C) ISO 9126 Quality Factors

It is an international standard software quality model provides help in creating a solid framework for assessing software. A standard software quality model can be calculated using following four ways -

1. Quality Model
2. External Metrics (those are applicable to running software)
3. Internal Metrics (those which do not rely on software execution (static measure)
4. Quality in use Metrics (only available when the final product is used in real conditions)

In this model, software product quality attributes were classified in a hierarchical tree structure of characteristics and sub characteristics.

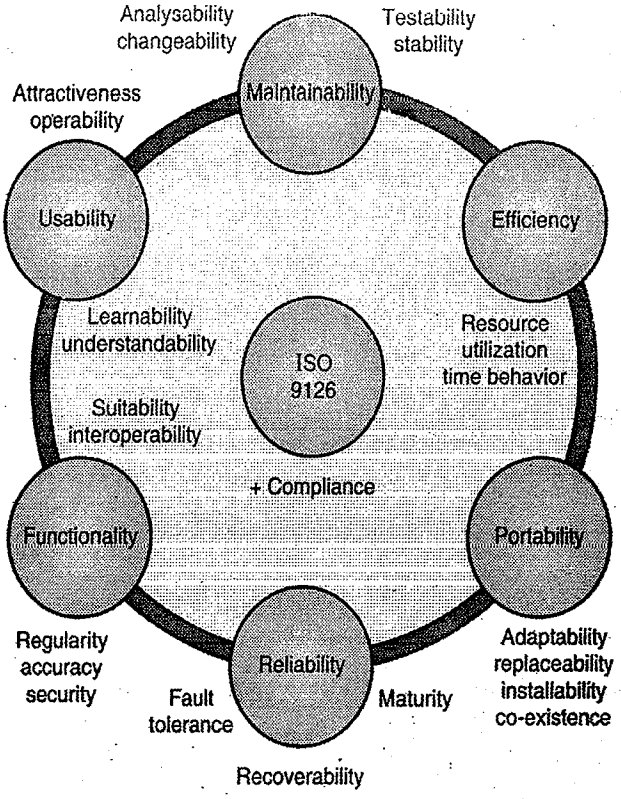


Fig. 5.1.4 : Quality criteria by ISO 9126

Highest level consists of the quality characteristics and the lowest level consists of the software quality criteria. The model specified six characteristics, which are further divided into 21 sub characteristics, which are further divided into attributes. An attribute is a measurable entity of the software product. Attributes are not defined in the standard, because they are varying between different software products.

Table 5.1.4 : ISO 9126 Quality attributes

1	Functionality	a set of software attributes with specific properties that provide functions that satisfy the needs of the user	(i) Suitability (ii) Accuracy (iii) Interoperability (iv) Security (v) Functionality compliance
2	Reliability	A set of software attributes with ability to maintain its specific level of performance under the specific stated conditions for a stated period of time.	(i) Maturity (ii) Fault tolerance (iii) Recoverability (iv) Reliability compliance
3	Usability	A set of software attributes that are measure of the effort needed user to learn to use the product.	(i) Understandability (ii) Learnability (iii) Operability (iv) Attractiveness (v) Usability compliance
4	Efficiency	A set of software attributes that represents the abllity of the software product to provide relationship between level of performance of the software and the amount of resources that are used under the stated conditions.	(i) Time behavior (ii) Resource utilization (iii) Efficiency compliance
5	Maintainability	A set of software attributes that are needed to avoid unexpected effects from specified modifications. This characteristic describes the ease with which the software product can be changed.	(i) Analyzability (ii) Changeability (iii) Stability (iv) Testability (v) Maintainability compliance
6	Portability	A set of software attributes that are needed for software to be transferred from one environment to another. This is important when the application is made for using on different distributed platforms.	(i) Adaptability (ii) Installability (iii) Co-existence (iv) Replaceability (v) Portability compliance

5.1.1(D) Targeted Quality Factors

The quality dimensions and factors are giving generic indication of the application's quality.

A software team develops a set of quality characteristics and associated questions that would examine the degree to which each factor has been satisfied.

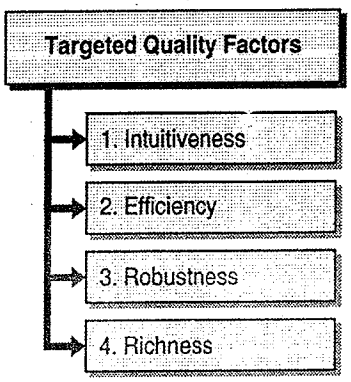


Fig. 5.1.5 : Targeted Quality Factors

➜ **1. Intuitiveness**

The degree to which expected usage patterns is followed by interface that can be used by a beginner without significant training. The interface should focus on - layout with easy usage and understanding, significant input economics (number of clicks and keystrokes) etc.

➜ **2. Efficiency**

The degree to which operations and associated information can be easily located and initiated. Interface should focus on - Good layout style allowing locating information and operation very efficiently, performing sequence of operation with significant amount of motion, easy to understand output representation, minimized depth of navigation for hierarchical operations.

➜ **3. Robustness**

The degree to which the software application handles bad input data or inappropriate user interaction. The attribute is focusing on - software ability to notice out of bound input, working without failure, recognizing common mistakes and guide user to back on right track, guiding during occurrence of error condition etc.

➜ **4. Richness**

The degree to which the application interface provides a set of rich features. The attribute is focusing on - customization of interface as per user need, ability of interface to enable user to identify sequence of common operations with single action.

Syllabus Topic : Software Quality Dilemma

5.1.2 Software Quality Dilemma

- According to Bertrand Meyer quality dilemma is as follows. People will reject the software that has terrible quality, It will be a complete lose because no one will want to buy it. Whereas if you spend more time, large effort, and huge investment to build the absolutely perfect software system, then it's going to be a long duration and expensive process that can take someone out of business.
- Due to this one may miss the market window, or simply exhaust all available resources.

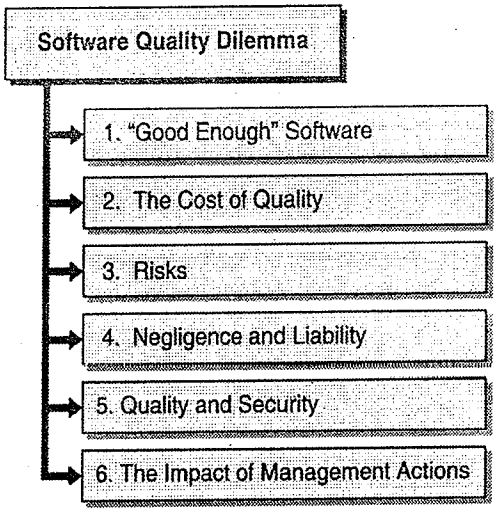


Fig. 5.1.6 : Software Quality Dilemma

➜ **1. "Good Enough" Software**

- An end user receives all desired high quality functions and features through a good enough software system. At the same time it also delivers other more specialized functions and features that has known bugs.
- In few application domains and for major software companies, this "good enough" may work. If a company with large marketing budget and can convince people to buy version 1.0, it is successfully locking them for reasonable period. This will not work for small companies.
- In case a "good enough" or a buggy product is delivered, then there is a risk of permanent damage to company's reputation. It will not give a chance to deliver version 2.0 because bad reputation may stop sales leads to shutting down of company.

- In real time system domains such as embedded software, automation software, telecommunication software etc., delivering good enough software may cost very high and may lead to criminal offense. Thus, good enough software only work in limited domains if delivered with greater cautions.

➜ **2. The Cost of Quality**

(i) Prevention cost

- Cost of planning and managing all quality control and quality assurance activities.
- Cost of training about all such activities.
- Cost of test planning
- Cost associated with technical activities related to requirement and design model.

(ii) Appraisal cost

- Cost for conducting technical reviews of work products
- Cost for data collection and metrics evaluation
- Cost for testing and debugging

(iii) Failure costs : Cost incurred before and after shipping the product to a customer.

(A) Internal Failure : Cost incurred when you detect an error in a product prior to shipment. It includes cost for rework, cost for finding defects in reworked product, cost for assessing modes of failure etc.

(B) External Failure : Cost incurred when you detect an error in a product after its shipment to customer. It includes cost for helpline support, warranty work, product return or replacement, complaint resolution etc.

➜ **3. Risks**

- People relies on software systems for their job, comfort, safety, entertainment, decision and entire life.
- Thus poor software quality can lead to high risk associated with it. Risk assessment and management must be a prime concern.

➜ **4. Negligence and Liability**

- Consider a scenario, where a company is hiring proficient developer or another company for requirement analysis, then for design and at last for implementing the system. In the beginning everything goes in smoother way, and it becomes worse once system is delivered.
- Common issues can be delay in delivery, non supported features, error in functionality etc. Customer blames developer for not following the given requirements and denies the payment. On the other side, developer is blaming customer for frequent change in requirements.

➜ **5. Quality and Security**

- The security of web applications is an important concern. Low quality software systems are easier to hack. Software security is completely related to quality.
- During every phase of SDLC one must think about reliability, availability, dependability, and security. Its better to identify problem in early stages.
- Bug is referred as implementation problem whereas flaw is known for architectural problem. Apply standard methods of design to avoid bugs and flaws.

➜ **6. The Impact of Management Actions**

- There is high impact of management decision on quality of software. Poor business decisions can be a disaster for an organization.
- **Estimation decisions :** Once delivery dates and budget is specified, team is conducting sanity check to ensure mapping of delivery dates and milestones. Due to market pressure team has to accept unrealistic delivery dates that leads to certain actions like, taking shortcuts, skipping good quality practices, providing subset of functionality etc.
- **Scheduling decisions :** Once project schedule is decided, sequence of activities to be performed is also formed based on the dependencies of various modules involved. During nearing deadline particular module need to be tested and its dependent are running slightly behind will introduce the hidden defects.
- **Risk oriented decisions :** A key attribute of successful software project is nothing but risk management. Development teams must be ready with recovery plan if anything goes wrong.

Syllabus Topic : Achieving Software Quality

5.1.3 Achieving Software Quality

- Software quality is the result of good project management and solid software engineering practice.
- Management and practice are applied within the context of following four activities which will help a software team achieve high software quality.

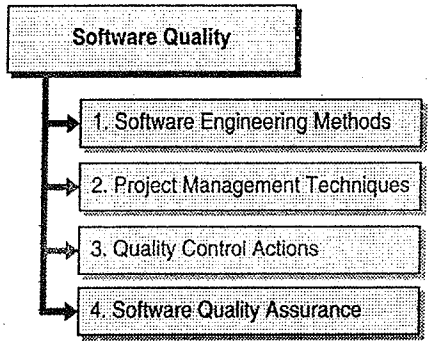


Fig. 5.1.7 : Software Quality

➔ **1. Software Engineering Methods**

- Understand the problem in order to build high quality software. The design that conforms to identified problem must be created. The software must have quality factors and characteristics (Refer to McCall's Quality Factors). By applying appropriate problem analysis and design methods one can build high quality software.

➔ **2. Project Management Techniques**

- Project manager should use estimation to check whether the delivery dates are achievable or not.
- Understand the dependencies while preparing for schedule to avoid shortcuts in development.
- Plan and management of risk is important factor to avoid negative impact on software quality.

➔ **3. Quality Control Actions**

- It consists of set of software engineering actions that ensures achievement of quality by each product.
- Models are reviewed for their completeness and consistency. Code inspection is done to detect and correct errors before the beginning of testing.
- Errors from processing logic, data manipulation, interface communication etc. are uncovered by applying series of testing.
- In case of failure in any work product, software team uses measurement and feedback technique to correct development process.

➔ **4. Software Quality Assurance**

- It covers software engineering methods, project management, and quality control actions that are necessary for building high-quality software.
- It performs assessment of effectiveness and completeness of quality control functions. For this it uses auditing and reporting techniques.
- It provides necessary data to management and technical staff about product quality that highlights working nature of actions taken for achieving quality. Management will take required action on the problems addressed by quality assurance.

Syllabus Topic : Software Quality Assurance (SQA)

5.2 Software Quality Assurance (SQA)

It is a set of activities for ensuring quality in software engineering processes (that ultimately result in the quality of software products). SQA is organized into goals, commitments, abilities, activities, measurements, and verification.

Some standard definitions of SQA are as follows -

- "A planned and systematic approach to the evaluation of the quality of and adherence to the software product standards, processes and procedures".

- "The function of software quality that assures that the standards, processes, and procedures are appropriate for the project and are correctly implemented."
- SQA focuses on two major points which are as follows -
  1. SQA includes all necessary activities that can contribute to the quality of software during the entire life cycle of the project.
  2. The emphasis must be on developing and systematically executing the Plan to achieve the objectives of software quality.

Process of the SQA

1. Requirement definition for fault detection, recovery and isolation of software system.
2. Take a periodic review to prevent software error and reduce functionality state of software development process and software product.
3. Write the process for defect measurement and analyzing defects as well as reliability and maintainability factors.

SQA Objectives

- Objective of SQA is to provide assurance that the procedures, tools, and techniques used during product development and modification are adequate.
- The following are the SQA objectives are as follows :
  - To provide approaches for quality management.
  - To provide mechanism for measuring and reporting defects.
  - To provide efficient and effective software-engineering process.
  - To provide a procedure to assure compliance with software-development standards .
  - To provide multi-testing strategies.
  - To take a formal technical reviews throughout the software process.

Syllabus Topic : SQA Elements

5.2.1 SQA Elements

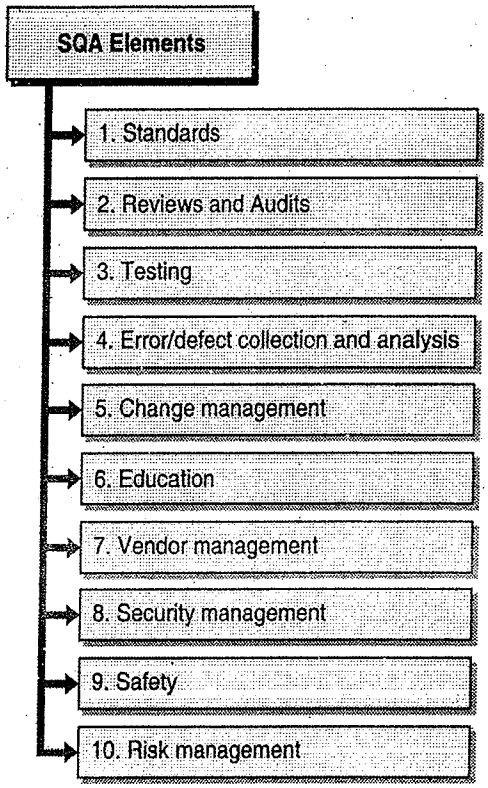


Fig. 5.2.1 : SQA Elements

1. Standards

The IEEE, ISO etc. have produced various software engineering standards and related documents. These standards may be adopted by a software engineering organization (voluntarily) or imposed by the customer/stakeholder. SQA will ensure that adopted standards are followed and all work products conform to them.

2. Reviews and Audits

Technical reviews are conducted by software engineers for software engineers to uncover errors. It is a quality control activity. Audit is a type of review conducted to ensure quality guidelines are being followed for software engineering work. It is conducted by SQA team.

3. Testing

Software testing carried out to find errors. SQA team will ensure that it is properly planned and conducted.

➜ **4. Error/defect collection and analysis**

SQA will collect and analyze error/defect data and find the root cause of it. SQA suggests actions to eliminate it.

➜ **5. Change management**

If change is not managed properly leads to confusion and ultimately to poor quality. SQA ensure that sufficient change management activities are deployed in organization.

➜ **6. Education**

It is a key aspect in improving software engineering practices. Engineers, managers, stakeholders etc. must be trained/educated through education programs. SQA takes this initiative and helps to improve software processes.

➜ **7. Vendor management**

SQA is suggesting specific quality practices that the vendor should follow may be through contracts.

➜ **8. Security management**

SQA ensures that software security is deployed through appropriate process and technology.

➜ **9. Safety**

The impact of hidden defects can be very dangerous. SQA assess the impact of software failure and initiate steps required to reduce risk.

➜ **10. Risk management**

SQA does the assessment of conduction of risk management activities. It checks plans for eliminating risks.

Syllabus Topic : SQA Tasks

5.2.2 SQA Tasks

The following are the main SQA tasks :

1. Prepare SQA plan for the project.
2. Participate in the development of the project's software process description.
3. Review software engineering activities to verify compliance with the defined software process.
4. Audit designated software work products to verify compliance with those defined as part of the software process.
5. Ensure that any deviations in software or work products are documented and handled according to a documented procedure.
6. Record any evidence of noncompliance and reports them to management.

Table 5.2.1 : SQA goals and attributes

Sr. No.	SQA Goal	Goal Attributes
1	Requirements quality	1. Ambiguity 2. Completeness 3. Volatility 4. Traceability 5. Model clarity
2	Design quality	1. Architectural integrity 2. Component completeness 3. Interface complexity 4. Patterns
3	Code quality	1. Complexity 2. Maintainability 3. Understandability 4. Reusability 5. Documentation
4	Quality control effectiveness	1. Resource allocation 2. Completion rate 3. Review effectiveness 4. Testing effectiveness

Syllabus Topic : SQA Goals and Metrics

5.2.3 SQA Goals and Metrics

The SQA goals are as follows :

– Perform as a planned and time bounded activity.

- Ensure the correctness, completeness, and consistency of the requirement model. It will have a strong influence on the quality of all work products that follow the model.
- Perform the assessment of every design model element to ensure it's quality and it's conformance to the requirements.
- Ensure the conformance of source code and related work products to local coding standards. Checking of attribute of source code's maintainability.
- A software team should apply available limited resources and achieve a high quality result. To report non-compliance issues that cannot be resolved in development are addressed by senior management.
- Stick on software products and activities to the applicable standards, procedures for software product.
- Verifies requirements objectively.
- To inform SQA activities and results to affected groups and individuals.

Syllabus Topic : Formal Approaches of SQA

5.2.4 Formal Approaches of SQA

There is a need of having a more formal approach to software quality assurance is required.

Experts like Dijkstra, Linger, Mills, and Witt proofed the program correctness and tied these proofs to the use of structured programming concepts.

1. Assumes that a rigorous syntax and semantics can be defined for every programming language.
2. Allows the use of a rigorous approach to the specification of software requirements.
3. Applies mathematical proof of correctness techniques to demonstrate that a program conforms to its specification.

Syllabus Topic : Statistical Software Quality Assurance

5.2.5 Statistical SQA

It talks more about quantitative aspect of quality. Following are the steps.

1. Collect and categorize information about software errors and defects.
2. Tracing the error and defect to its roots. e.g., non-conformance to the requirement specifications, error in design,, non use of standards, communication gap with the customer.
3. Use the Pareto principle i.e. 80% of the defects can be traced to 20% of all possible causes,and isolate the 20% that are vital.
4. After identifying cause of vital few, correct the occurred problems that have caused the errors and defects.

Syllabus Topic : Six-Sigma for Software Engineering

5.2.6 Six-Sigma

- It is referred as the strategy for statistical quality assurance. Six sigma as a business management strategy which is always aiming at improving the quality of processes.
- Six sigma does this by minimizing and eventually removing the errors and variations. Motorola introduced the concept in the 1980s as, "It is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects' in manufacturing and service-related processes".
- A term Sigma (Greek term) used in statistics to represent standard deviation from mean value, an indicator of the degree of variation in a set of a process.
- Sigma measures how far a given process deviates from perfection.
- Higher sigma capability, better performance. The term Six Sigma is derived from six standard deviations-3. 4 instances (defects) per million occurrences implying an extremely high quality standard.

Sigma Levels

Sigma measurements against defect rates is shown in Table 5.2.2. No process can deliver 100% quality at a time. Six is highest sigma rating indicates that product is meeting all time and quality criteria of customer.

Table 5.2.2 : Sigma measurement

Sigma Level / Process Capability	Defects Rate	On-time Quality Rate
0	93.32 %	6.68 %
1	69.15 %	30.85 %
2	30.85 %	69.15 %
3	6.68 %	93.32 %
4	0.62 %	99.38 %
5	0.23 %	99.77 %
6	0.034 %	99.966 %

☞ Three core steps of Six Sigma

1. Define customer requirements and deliverable, and project goals through a well defined method of customer communication.
2. Measure the existing process along with its output in order to determine current quality performance (collect defect metrics).
3. Analyze defect metrics and determine the important problems, causes, and effects

☞ Six Sigma Methodology

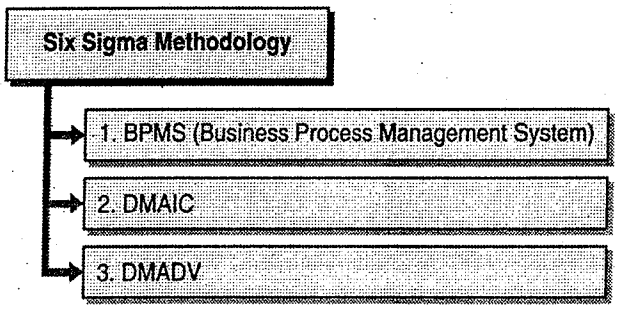


Fig. 5.2.2 : Six Sigma Methodology

➜ 1. BPMS (Business Process Management System)

– It emphasize on process improvement and automation in order to derive performance. Combination of six sigma and BPM strategies forms a powerful way to improve performance. However, both strategies are not mutually exclusive but some companies have produced good results by combining them.

➜ 2. DMAIC

– It is also referred as Six Sigma Improvement Methodology. It is a logical and structured approach to problem solving and process improvement. It is an iterative process (continuous improvement) and a quality tool which focus on change management style. It has total five core steps Define, Measure, Analyze, Improve, Control.

– The first three core steps remains common. If an existing software process is in place, but improvement is required, Six Sigma suggests two additional steps :

☞ Steps of Six Sigma

(i) Define (the project goals and customer's internal and external deliverable)

(ii) Measure (the process to determine current performance; quantify the problem)

(iii) Analyze (Analyze and determine the root causes of the defects)

(iv) Improve the process by eliminating the root causes of defects, on bases of measurements and analysis

(v) Control the process to ensure that future work does not reintroduce the causes of defects.

Table 5.2.3 : Six Sigma DMAIC Framework and toolkit

Step	Explanation	Toolkits used
Define	Define project goals, benefits, internal and external customer deliverable, the business problem, resources, project scope and high-level project boundaries, the customer(s), Voice of the customer (VOC), Critical to Quality (CTQs), process outputs, metrics. Project charter document captures this information.	1. Charter 2. Voice of Customer 3. Baseline 4. Benchmark

Step	Explanation	Toolkits used
Measure	Define detailed process map for recording the activities performed as part of a process. Determine Process Capability by comparing current process performance baselines with performance metric at the end of project. This will give the required improvement. It collects data from a large number of sources to determine types of metrics and defects.	1. Seven basic QA tools 2. Data collection methods 3. Sampling techniques
Analyze	Once process performance has been quantified, the analyze phase helps identify, validate and select root possible causes of the problems. Root-cause analysis can be done using fishbone diagram.	1. Root-cause analysis 2. Control charts 3. Reliability analysis 4. Effect analysis
Improve	Identify, test and implement a solution to the problem in order to achieve the desired breakthrough performance. Design innovative and creative solutions to fix and prevent process problems.	1. Design and modeling
Control	Objective of this step is to sustain the improvements. Create a control plan to keep an improved process at its current level. Implement Statistical Process Control for monitoring process behavior and develop a response plan in case performance variation. A Control chart is used to assess the stability of the improvements	1. Control charts 2. Performance management

➜ 3. DMADV

It is also referred as creating new process which will perform at Six Sigma. It is applicable if an organization is developing a software process rather than improving an existing process. It consists of five core steps as define, measure, analyze, design, and verify; out of which first three steps are common.

(i) Define (project definition)

(ii) Measure (measure the opportunity measurement)

(iii) Analyze (process options analysis)

(iv) Design a process to avoid root causes of defects and to meet customer requirements.

(v) Verify performance (avoiding defects and meeting customer requirements).

☞ Benefits of Six Sigma

- Improving customer satisfaction by reducing defects thus improves productivity.
- Helps to set performance goal for everyone.
- Enhancing the value for customers.
- Accelerating the rate of improvement.
- Promoting learning across boundaries.
- Executes strategic change.
- Syllabus Topic : Six-Sigma for Software Engineering

Syllabus Topic : ISO-9000 Quality Standards

5.2.7 ISO-9000 Standard

- International Organization for Standardization's (ISO) 9000 is the set of standards related to software quality. It is concerned with quality management and quality assurance.
- The ISO 9000 standards helps the organizations to ensure that they are meeting the needs of customers and other stakeholders. In this process organizations also have to meet statutory and regulatory requirements related to a product or service.

- ISO 9000 deals with the fundamentals of quality management systems. ISO 9001 deals with the requirements that organizations wishing to meet the standard must fulfill. These standards are applicable to areas (but not limited) such as - government, education, banking, telecommunication, software development, agriculture, manufacturing etc.

History and Revisions

- Different technical committees and advisory groups are continuously revising the ISO 9000 standard based on feedback received from those who are implementing this standard.
- International Organization for Standardization (ISO) published the standard in 1987.
- It is a specialized international agency for standardization composed of the national standards bodies of more than 160 countries.
- Major revision of standard was made in 1994 and 2000; then it was revised again in 2008. In September 2015 current versions of ISO 9000 and ISO 9001 were published.

ISO 9000 Family

The phrase "ISO 9000 family" or "ISO 9000 series" refers to a group of quality management standards which are process standards and not a product standards.

- **ISO 9000:2015** - Quality management systems – Fundamentals and Vocabulary (definitions), referenced in all ISO 9000 Standards.
- **ISO 9001:2015** - Quality management systems – Requirements, those must be complied by an organization to become ISO 9001 certified.
- **ISO 9004:2009** - Quality management systems - Managing for the sustained success of an organization. It provides guidelines for sustaining QMS (Quality management systems) success through evaluation and continuous performance improvement.
- **ISO 19011:2011** - Guidelines for auditing management systems.

Quality Management Principles

The ISO 9000 series family of standards are based on seven quality management principles (QMP). The seven quality management principles are :

Table 5.2.4 : Quality management principles

QMP 1 - Customer focus	Organizations should understand current and future needs of customer, focus on meeting customer requirements, and aim to exceed customer expectations.
QMP 2 - Leadership	Leaders are responsible to establish unity in an organization. By giving proper directions, they should create and maintain the internal environment where people are involved to achieve the organization's objectives.
QMP 3 - Engagement of people	All the people of an organization should be fully involved which will enable utilization of their abilities for the organization's benefit.
QMP 4 - Process approach	Manage all the activities and related resources as a process that will bring desired result.
QMP 5 – Improvement	Organization objective should be overall performance improvement.
QMP 6 - Evidence-based decision making	Analysis of data and information leads to effective decisions.
QMP 7 - Relationship management	An organization and its external providers such as, suppliers, contractors, service providers etc. are interdependent with each other, hence a mutually beneficial relationship should be established.

ISO Registration

A formal audit of 20 elements in involved to obtain ISO registration. The outcome of all these elements need to be positive. Following are the elements :

1. Management responsibility
2. Quality system
3. Contract review
4. Design control

5. Document control
6. Purchasing
7. Purchase-supplier product
8. Product identification and traceability
9. Process control
10. Inspection and testing
11. Inspection, measuring, and test equipment
12. Inspection and test status
13. Control of non-conforming product
14. Corrective action
15. Handling, storage, packaging, and delivery
16. Quality records
17. Internal quality audits
18. Training
19. Servicing
20. Statistical techniques

☞ Software Metrics Requirements

(a) Product Metrics

- It includes metrics such as, defects from customers point of view, reported field failures etc. It is required to collect metric data and report its values.
- On the basis of this data, identify the current level of performance of each metric. Based on performance define specific improvement goals.

(b) Process Metrics

- It is required to check the compliance of in process quality objectives.
- It checks how effectively the development process is being carried out by seeing reduction in the probability of undetected faults or introduction of new faults.

☞ Need of ISO Certification

- Having ISO certificate is a sign of customer confidence and in turn becomes motivating factor for organizations.
- ISO certification indirectly provides standard for international bidding.
- Processes becomes more focused, cost-effective and efficient.
- It enables design of high-quality repeatable software products.
- Weaknesses are highlighted and corrective measures for improvements are suggested.
- It facilitates the process development and total quality measurement.
- More weightage is given to the need for proper documentation.

☞ Advantages of ISO Standard

- Proper quality management can improve and increase business as it puts measure of customer needs and satisfaction first.
- It has a positive effect on investment, competition, sales growth and margin, market share.
- Affected stakeholders and their expectations can be easily determined by the organizations.
- Highlights business objectives and new business opportunities.
- Enables organizations to identify and address the risks associated.
- Work efficiency is increased because of all their processes are aligned and understood by everyone.
- Increased productivity and efficiency, bringing internal costs down.
- Helps to meet necessary statutory and regulatory requirements.
- With ISO 9001 standard, organizations can expand into new markets/sectors.

☞ Limitations of ISO Standard

- It misleads the companies into thinking that certification means better quality.
- High amount of money, time, and paperwork required for registration, increased bureaucratic processes.

- No idea about, what kind and how many resources will be needed, How much will be the cost of certification.
- Risks and uncertainty of not knowing about the direct relationships to improved quality.
- Failure to get certified has a risk of poor company image.
- ISO 9001 specifications do not guarantee a successful quality system.
- Certifications are in fact based on customer contractual requirements rather than a desire to actually improve quality.
- Measurement of, processes or parameters ensuring quality is not carried out.
- ISO does not validate technical solution required for advanced quality planning.

Syllabus Topic : SQA Plan

5.2.8 SQA Plan

SQA has following plans

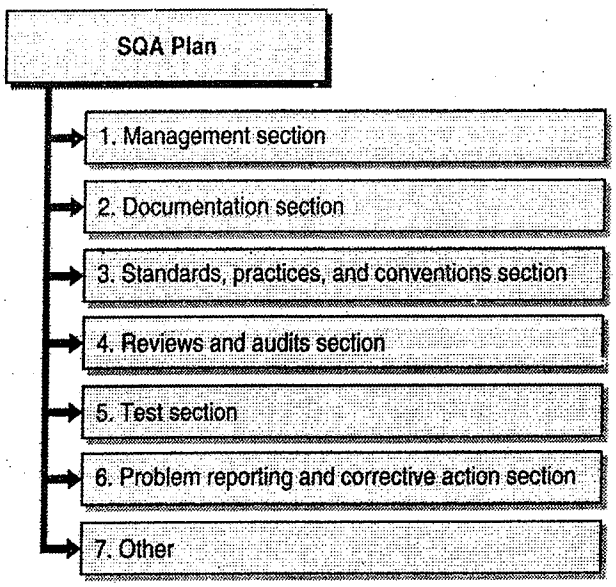


Fig. 5.2.3 : SQA plan

➜ **1. Management section**

It describes the place of SQA in the structure of the organization.

➜ **2. Documentation section**

It describes each work product produced as part of the software process.

➜ **3. Standards, practices and conventions section**

It lists all applicable standards/practices applied during the software process and any metrics to be collected as part of the software engineering work.

➜ **4. Reviews and audits section**

It provides an overview of the approach used in the reviews and audits to be conducted during the project.

➜ **5. Test section**

It gives references the test plan and procedure document and defines test record keeping requirements.

➜ **6. Problem reporting and corrective action section**

This defines procedures for reporting, tracking, and resolving errors or defects, identifies organizational responsibilities for these activities.

➜ **7. Other**

It consists of tools, SQA methods, change control, record keeping, training, and risk management.

Review Questions

Q. Explain software quality in brief. **(Section 5.1.1)**

Q. What are the objectives of software quality assurance? **(Section 5.2)**

Q. Describe Software Quality Assurance plan. **(Section 5.2.8)**

Q. Briefly explain the ISO 9126 quality factors. **(Section 5.1.1(C))**

Q. What is the role of an SQA group? **(Section 5.2)**

Q. State McCall's quality factors **(Section 5.1.1(B))**

Q. What steps are required to perform statistical SQA? **(Section 5.2.5)**

Q. Describe various Six Sigma Methodologies. **(Section 5.2.6)**

Q. What is DMAIC? **(Section 5.2.6)**

Q. Discuss ISO 9000 standard. **(Section 5.2.7)**

Q. What is ISO standard? What are its advantages? **(Section 5.2.7)**

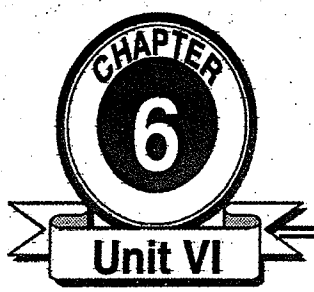
Q. Write in brief about ISO 9000 family of standards. **(Section 5.2.7)**

Q. What are the different quality dimensions suggested by Garvin? **(Section 5.1.1(A))**

Q. How cost and risk factors are affecting software quality? **(Section 5.1.2)**

Q. What are the different Software Quality Assurance elements? Explain in brief. **(Section 5.2.1)**

❑❑❑



Software Quality Tools

Syllabus Topics

Total Quality Management, Product Quality Metrics, In process Quality Metrics, Software maintenance, Ishikawa's 7 basic tools, Checklists, Pareto diagrams, Histogram, Run Charts, Scatter diagrams, Control chart, Cause Effect diagram, Defect Removal Effectiveness and Process Maturity Level.

Syllabus Topic : Total Quality Management

6.1 Total Project Quality Management

- The term Total Quality Management (TQM) was originally coined by the Naval Air Systems Command to describe its Japanese-style management approach to quality improvement in 1985.
- It is also called as **Total Quality Control (TQC).** It shows the way of managing organization to achieve excellence. Total means everything, Quality means degree of excellence, and Management means art, actor way of organizing, controlling, planning, directing to achieve certain goals.
- Therefore, TQM is considered to be an art of managing the whole to achieve excellence. Total Quality Management means the organization's culture is defined and supports the constant attainment of customer satisfaction with an integrated system of tools, techniques, and training.
- General TQM approach model is shown in Fig. 6.1.1.

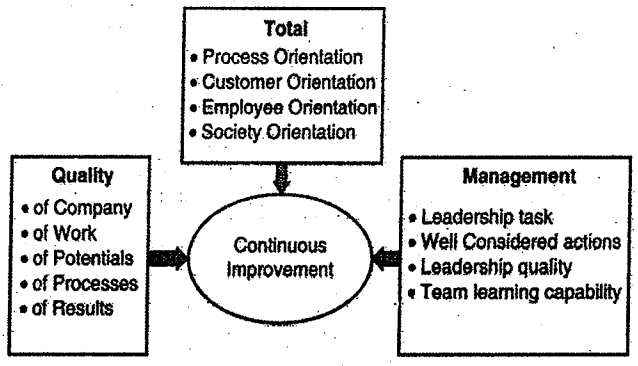


Fig. 6.1.1 : TQM Approach

- TQM approach involves the continuous improvement of organizational processes, that will always result in high quality products and services.
- The goal is customer satisfaction. Through TQM continuous efforts are directed by the management as well as employees of a particular organization to ensure long term customer loyalty and customer satisfaction.
- TQM ensures long term success, since it makes every single employee to work towards the improvement of work culture, processes, services, systems.

– The effects of TQM is illustrated in Fig. 6.1.2.

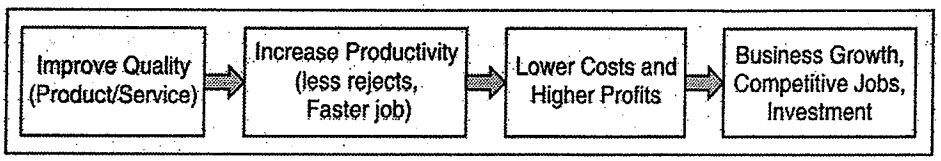


Fig. 6.1.2 : Effects of TQM

6.1.1 Key Elements of a TQM System

1. **Customer focus :** The main objective is to attain total customer satisfaction. Customer focus includes tasks like, studying customer's needs, gathering customers' requirements, measuring customer satisfaction, and managing customers satisfaction.
2. **Process :** The main objective is to achieve continuous process improvement by reducing process variations. It includes both business process and product development process. Product quality can be enhanced through process improvement.
3. **Human side of quality :** The main objective is to create a company-wide quality culture. It focuses on areas like, leadership, management commitment, total participation, employee empowerment, and other social, psychological, and human factors.
4. **Measurement and analysis :** The main objective is to drive continuous improvement in all quality factors by the goal-oriented measurement system.

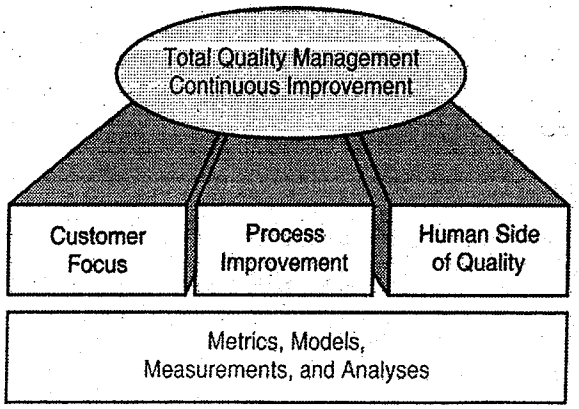


Fig. 6.1.3 : Key elements of TQM

6.1.2 Other Elements of TQM

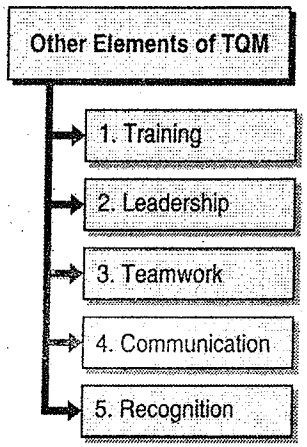


Fig. 6.1.4 : Other elements of TQM

➜ **1. Training :** Leaders and managers who lead the team should make their team members aware of the benefits and procedures of Total Quality Management. The team must also be trained on interpersonal skills, technical skills, problem solving skills, decision making skills, and the like.

➜ **2. Leadership :** With the right leaders providing a direction to the employees to work towards Total Quality Management, it can prove to be a strong source of inspiration for the team members to tread the path. Without a properly defined direction, individuals may take their own individual paths, which may lead everything to go haywire.

➜ **3. Teamwork :** Teamwork is one of the most crucial and unavoidable elements of Total Quality Management. When united, employees can combine various ideas to come up with a great brainstorming procedure that can lead to amazing improvisations and results.

➜ **4. Communication :** It is obvious that no team can be a success without effective communication. It is communications that binds employees together and bring out the best from them. Lack of communication of information could lead to misunderstandings, and in turn problems.

➜ **5. Recognition :** It drives employees to work hard and deliver their best.

6.1.3 TQM Frameworks

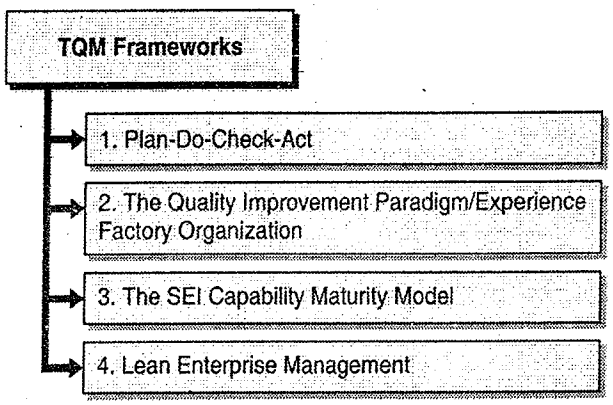


Fig. 6.1.5 : TQM Frameworks

➜ **1. Plan-Do-Check-Act**

- It is also known as the **Deming cycle or PDCA cycle.** It is a four step management method which work in an iterative fashion.
- It works on the basis of a feedback cycle that is used for optimizing a single process or entire production line.
- It uses feedback loops and statistical quality control techniques to experiment with methods for improvement and to build predictive models of the product.

➜ **2. The Quality Improvement Paradigm/Experience Factory Organization**

- Main purpose is to build a continuously improving organization. This is based on evolving goals and assessment of those goals.
- An internal assessment is done against the organization's own goals and status (rather than process areas).
- By making use of techniques such as qualitative analysis, quantitative analysis, Goal-Question-Metric (GQM), model building etc. Products are improved through the process.

➜ **3. The SEI Capability Maturity Model**

- It is a staged process improvement, based on assessment of key process areas, represents a continuous process improvement.
- It is a five-level process maturity model is defined based on repeated assessments of an organization's capability in key process areas. Improvement is achieved by action plans for poor process areas.

➜ **4. Lean Enterprise Management**

- It is based on the principle of concentration of production on "value-added" activities and the elimination or reduction of "not-value-added" activities.
- The goal is to build software with the minimum necessary set of activities and then to tailor the process to the product's requirements.
- The approach uses such concepts as technology management, human-centered management, decentralized organization, quality management, supplier and customer integration, and internationalization.

6.1.4 Importance of TQM

- Ensures superior quality products and services.
- Essential for customer satisfaction which eventually leads to customer loyalty.
- Tools help an organization to design and create a product which the customer actually wants and desires.
- Ensures increased revenues and higher productivity for the organization.
- Helps organizations to reduce waste and inventory.

6.1.5 Models of TQM

1. ISO quality management standards
2. Deming Application Prize
3. European Foundation for Quality Management, and
4. Malcolm Baldrige Criteria for Performance Excellence

6.1.6 Software Quality Metrics

- It is a subset of software metrics that focus on the various quality aspects of the product, process, and project. It is closely associated with process and product metrics as compared to project metrics.
- Product metrics (describe the characteristics of the product) - size, complexity, design features, performance, and quality level.

- Process metrics (used to improve software development and maintenance) - effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process.
- Project metrics (describe the project characteristics and execution) - cost, schedule, number of software developers, productivity, the staffing pattern over the life cycle of the software.
- Another category of quality metrics is focusing on measurement of quality level of the maintenance process

Syllabus Topic : Product Quality Metrics

6.2 Product Quality Metrics

- Software quality has two levels : Intrinsic product quality and customer satisfaction. Following metrics cover both levels.
  1. Mean time to failure
  2. Defect density
  3. Customer problems
  4. Customer satisfaction
- Intrinsic product quality is measured based on count of bugs or functional defects in the software. This can be referred as defect density. Defect density gives defects relative to the software size such as lines of code, function points etc. Defect density is used in commercial software systems.
- Intrinsic product quality is also measured as how long the software system can run before it crashes. This can be referred as mean time to failure(MTTF). MTTF measures the time between failures. MTTF used in safety-critical systems.
- Separating two terms i.e. failure and defect in data tracking, actual measurements is very difficult. According to the IEEE/ American National Standards Institute (ANSI) standard (982.2) following are the definitions.
  - Error : Human mistake that results in incorrect software.
  - Fault : An accidental condition that causes a unit of the system to fail to function as required.
  - Defect : Anomaly in a product.
  - Failure : Occurs when a functional unit of a software-related system can no longer perform its required function or cannot perform it within specified limits.

6.2.1 The Defect Density Metric

- Many issues are involved in comparing the defect rates of software products. To define defect rate first define numerator, denominator, and specify the time frame.
- The numerator term can be number of defects in the software. The denominator term can be size of the software (in thousand lines of code (KLOC) or in the number of function points).

6.2.1(A) Lines of Code (LOC)

- A well known author Conte defined the term LOC in his literature named as **"Software Engineering Metrics and Models" as follows : 'A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements.'**
- LOC is simple measure but the differences between physical lines and instruction statements or logical lines of code along with differences among programming languages produces number of variations in counting LOCs. Few are as follows.
  - Count only executable lines.
  - Count executable lines plus data definitions.
  - Count executable lines, data definitions, and comments. Count executable lines, data definitions, comments, and job control language.
  - Count lines as physical lines on an input screen.
  - Count lines as terminated by logical delimiters.
- Newly released (first time) product quality (projected and actual) can be easily stated using certain specified LOC count method. But it is difficult to calculate defect rate for new and changed code. Following two things must be available.
  - **LOC count :** The entire software product, new and changed code of the release.

- **Defect tracking :** Defects must be tracked to the release origin. It is the portion of the code that contains the defects and at what release this portion was added, changed, or enhanced. To calculate defect rate of the entire product use all defects. To calculate defect rate for the new and changed code use only defects of the release origin of the new and changed code.

☞ **Example: LOC Defect Rates**

- At IBM Rochester, LOC data depends on instruction statements (logical lines of codes). It includes executable code, data definitions and excludes comments. LOC counts are derived for the total product and for new and changed code of new release. The two size metrics defined as shipped source instructions (SSI) and new/changed source instructions (CSI). These two metrics are defined because the LOC count is based on source instructions. Following formula depicts relationship between the SSI count and the CSI count:
- SSI (current release) = SSI (previous release) + CSI (new and changed code instructions for current release) - Deleted code (very small) - Changed code (to avoid double count in both SSI and CSI)
- The defects tracked after the release of the product can be field defects, which are found by customers, or internal defects that found internally. Following are the post-release defect rate metrics per thousand SSI (KSSI) or per thousand CSI (KCSI):

(1) Total defects per KSSI (measures code quality of the total product).

(2) Field defects per KSSI (measures defect rate in the field).

(3) Release-origin defects (field and internal) per KCSI (measures development quality).

(4) Release-origin field defects per KCSI (measures development quality per defects found by customers).

- Metric-1 measures the total release code quality, and metric-3 measures the quality of the new and changed code. Both metrics are same for the initial release where the entire product is new. After progressing, Metric-1 is affected by aging/improvement/deterioration of Metric-3. Both are process measures; whereas Metric-2 and Metric-4 represents customer's perspective.

6.2.1(B) Customer's Perspective

- It's a good practice to consider customer perspective in quality software engineering. Defect rate metrics measure code quality per unit and is useful to drive quality improvement from developer point of view.
- Assume the setting of defect rate goal for release-to-release improvement of a product. From the customer's point of view, the defect rate is just a number of defects that affect their business. Reducing total number of defects from a release-to-release (irrespective of size) is considered as good target. If new release is larger than the predecessors, that means defect rate goal for the new/changed code must be better than previous release in order to reduce the total number of defects.

☞ **Hypothetical example**

- **Initial Release of Product A**
  - KCSI = KSSI = 50 KLOC; Defects/KCSI = 2.0
  - Total no. of defects = 2.0 x 50 = 100
- **Second Release**
  - KCSI = 20
  - KSSI = 50 + 20 (new and changed LOC) - 4 (assuming 20% are change LOC) = 66
  - Defect/KCSI = 1.8 (assuming 10% improvement over the first release)
  - Total no. of additional defects = 1.8 x 20 = 36
- **Third Release**
  - KCSI = 30
  - KSSI = 66 + 30 (new and changed LOC) - 6 (assuming 20% of changed LOC) = 90
  - Targeted number of additional defects (no more than previous release) = 36
  - Defect rate target for the new and changed LOC: 36/30 = 1.2 defects/K or lower

- Improvement in defect rate by 10% from initial release to second release. Due to smaller second release, a 64% reduction [(100 - 36)/100] in the number of defects experienced by customer.
- The size of third release is larger than second, hence this factor works. The defect rate of third release has to be one-third (1.2/1.8); means better than that of the second release for the number of new defects not to exceed that of the second release.
- In case of large difference in two defect rates, other actions should be planned for base code quality improvement or to reduce the volume of post-release field defects by finding them internally.

6.2.1(C) Function Points

- It is an another method of measuring size of a software system. There are two ways of using function points in application development in terms - productivity (e.g., function points per person-year) and quality (e.g., defects per function point).
- According to Conte - "a function is a collection of executable statements performing a certain task, with declarations of the formal parameters and local variables manipulated by those statements"
- Software productivity is measured by knowing number of function developed by development team using given resources. Defect rate can be measured with respect to number of functions provided by software. The low value of defect rate per unit of function indicates high quality irrespective of number of LOC that function has.
- Function point method (by Albrecht in 1979 at IBM) is addressing problems related to LOC while measuring size and productivity.
- Function point (FP) is calculated by adding weights of below mentioned five major components that comprise an application:
  1. Number of External Inputs (EI) (e.g., transaction types) x weighting factors.
  2. Number of External Outputs (EO) (e.g., report types) x weighting factors.
  3. Number of Internal Logical Files (ILF) (files as the user might conceive them, not physical files) x weighting factors.
  4. Number of External Interface Files (EIF) (files accessed by the application but not maintained by it) x weighting factors.
  5. Number of External Inquiries (EQ) (types of online inquiries supported) x weighting factors.
- The weighting factors is selected based on complexity assessment of application. The complexity of each component depends on set of standards that define complexity in terms of objective guidelines. Following are the weighting factors choices.
  - **External input** : low complexity=3; high complexity=6; average complexity=4
  - **External output :** low complexity=4; high complexity=7; average complexity=5
  - **Internal logical file** : low complexity=7; high complexity=15; average complexity=10
  - **External interface file :** low complexity=5; high complexity=10; average complexity=7
  - **External inquiry :** low complexity=3; high complexity=6; average complexity=4
- First step is to calculate the function counts by using weighting factors given as follows:

$$FC = \sum_{i=1}^{5} \sum_{j=1}^{3} W_{ij} \times X_{ij}$$

  - W_{ij} - the weighting factors of the five components by complexity level (low, average, high).
  - X_{ij} - the numbers of each component in the application.
- In second step impact of 14 General System Characteristics (GSC) in terms of their likely effect on the application is assessed on a scale from 0 to 5.

The 14 GSCs are:

1. Data communications
2. Distributed functions
3. Performance

4. Heavily used configuration
5. Transaction rate
6. Online data entry
7. End-user efficiency
8. Online update
9. Complex processing
10. Reusability
11. Installation ease
12. Operational ease
13. Multiple sites
14. Facilitation of change

- The scores which are ranging from 0 to 5 for these characteristics are added and Value Adjustment Factor (VAF) is calculated as –

$$VAF = 0.65 + 0.01 \sum_{i=1}^{14} C_i$$

  - i is from 1 to 14 representing each GSC.
  - C_i is the score for each general system characteristic 'i'.

- Finally, number of FP (function points) is derived as:

$$FP = FC \times VAF$$

Example: Function Point Defect Rates

- The estimation for DRE (Defect Removal Efficiency) of software organizations by level CMM (Capability Maturity Model) developed by the Software Engineering Institute (SEI) is considered for example.
- By applying DRE to the overall defect rate per function point, the following defect rates for the delivered software were estimated.
- The time frames below defect rates is not specified but are from the maintenance life of the software.
- The estimated defect rates per function point are given below.
  - SEI CMM Level 1: 0.75
  - SEI CMM Level 2: 0.44
  - SEI CMM Level 3: 0.27
  - SEI CMM Level 4: 0.14
  - SEI CMM Level 5: 0.05

6.2.2 Customer Problems Metric

- The metric is used measure the problems customers encounter while using the product. These problems may be due to valid defects or non-defect-oriented problems (such as, usability problems, user errors, unclear documentation and information, duplicates of valid defects). The total number of problems in a software is a combined total of both above types of defects.
- Problem per user month (PUM) is the problem metric

  PUM = Total problems that customers reported (true defects and non-defect-oriented problems) for a time period + Total number of license-months of the software during the period

  Where,

Number of license months	=	Number of install licenses of the software	×	Number of months in the calculation period

- PUM is calculated monthly after release and also yearly (avg. of all months). Following are the three approaches to lower the value of PUM and thus improve the quality and achieve business goals of organization.
- Table 6.2.1 highlights key points related to defect rate metric and PUM metric.

Table 6.2.1 : Defect Rate and Customer Problems Metrics

Parameter	Defect Rate	Problems per User Month (PUM)
Numerator	Valid and unique product defects	All customer problems (defects and non-defects, first time and repeated)
Denominator	Size of product (KLOC or function point)	Customer usage of the product (user-months)
Measurement perspective	Producer software development organization	Customer
Scope	Intrinsic product quality	Intrinsic product quality plus other factors

- Customer metric lies in between defects measurement and customer satisfaction. To minimize customer problems - reduce the product functional defects, improve documentation, usability, problem rediscovery.

- To maximize customer satisfaction - reduce defects, manage schedule, total customer solution, availability of product, services company image.
- From the software quality standpoint, the relationship of three metrics - defect rate, customer problems, customer satisfaction, can be represented by Venn diagram as shown in Fig. 6.2.1.

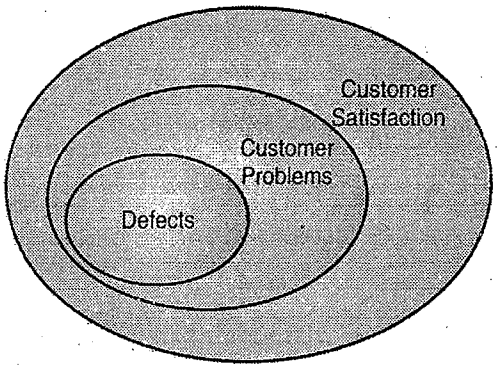


Fig. 6.2.1 : Scope of Three Quality Metrics

6.2.3 Customer Satisfaction Metrics

- Customer satisfaction measured through customer survey data via the five-point scale - Very satisfied, Satisfied, Neutral, Dissatisfied, Very dissatisfied. Using methods like customer surveys, satisfaction with overall product quality and some of its specific dimensions is derived.

☞ **Some of the well known examples**

1. **IBM :** CUPRIMDSO categories (capability, functionality, usability, performance, reliability, installability, maintainability, documentation/information, service, and overall).
2. **Hewlett-Packard :** FURPS (functionality, usability, reliability, performance, and service).

- Based on the five-point-scale data, following metrics with minor variations can be constructed and used, depending on the purpose of analysis.

☞ **Example**

(i) Percentage of completely satisfied customers.

(ii) Percentage of satisfied customers (satisfied and completely satisfied).

(iii) Percentage of dissatisfied customers (dissatisfied and completely dissatisfied).

(iv) Percentage of non-satisfied (neutral, dissatisfied, and completely dissatisfied).

- Some companies use NSI - Net Satisfaction Index, for comparisons across product. Following are the weighting factors used by NSI: Completely satisfied = 100%, Satisfied = 75%, Neutral = 50%, Dissatisfied = 25%, Completely dissatisfied = 0%
- NSI ranges from 0% i.e. all customers are completely dissatisfied to 100% i.e. all customers are completely satisfied. Value of 75% indicates all customers are satisfied but not completely satisfied.

Syllabus Topic : In Process Quality Metrics

6.3 In process Quality Metrics

- In-process quality metrics helps to understand programming process and to learn to engineer quality into the process. Following are the four in-process quality metrics.

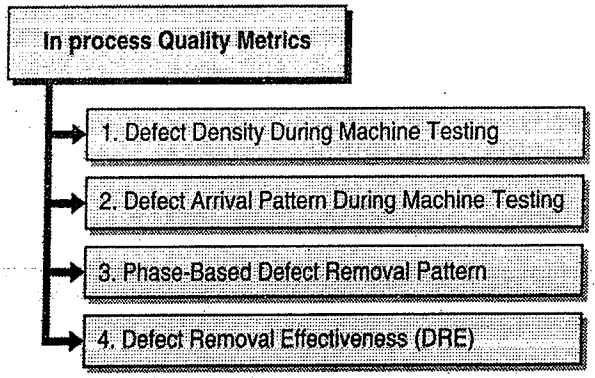


Fig. 6.3.1 : In process quality metrics

6.3.1 Defect Density During Machine Testing

- The formal machine testing is a type of testing done after code is integrated into the system library. Defect rate during machine testing is correlated with field defects.
- Higher defect rates found during testing indicates that higher error injection is experienced during software development process. If a product or module has higher testing defects, it is due to more effective testing or higher hidden defects in the code. Principle of testing says that the more defects found during testing, the more defects will be found later.

- Quality of a software when it is still in testing, can be represented by defects per KLOC or per function point. This is helpful in monitoring subsequent releases of a product of same company.
- Following questionnaire can be used by project manager or development team to decide quality.
  (i) If rate of defect during testing is the same or lower than that of previous release (or a similar product), then ask: Does testing for current release deteriorate?
    - If no, it means positive quality perspective.
    - If yes, then need to do extra testing like add more test cases to increase coverage, stress testing, customer testing.

  (ii) If rate of defect during testing is substantially higher than previous release (or a similar product), then ask: is there anything planned for improving testing effectiveness?
    - If no, it means negative quality perspective. Generally the only remedial approach at this stage of the SDLC is to perform more testing, that will produce even higher defect rates.
    - If yes, it means positive quality perspective.

6.3.2 Defect Arrival Pattern During Machine Testing

- The defect arrival pattern (times between failures) gives more information compared to summarized information shown by defect density. There may different patterns of defect arrivals indicate different quality levels in the field for same overall defect rate during testing.
- Fig. 6.3.2 shows two contrasting patterns for defect arrival rate and summarized defect rate. Data plotted from 44 weeks before code-freeze until the week just before code-freeze. The second pattern uses charts on the right side indicates the late start of testing, insufficient test suite, and testing ended prematurely.

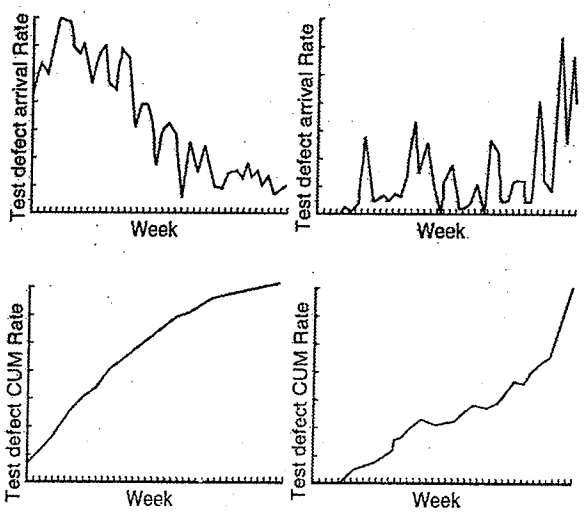


Fig. 6.3.2 : Two Contrasting Defect Arrival Patterns During Testing

- The objective must be, observe the defect arrivals to stabilize it at a very low level, or times between failures that are far apart. This need to be done before ending the testing effort and releasing the software to the field.
- Many software reliability models uses these declining patterns of defect arrival during testing. The defect arrival pattern is observed for usually weeks and occasionally months. Reliability models that needs execution time data may use time interval in units of CPU time.
- Following are three metrics that should be considered simultaneously :
  (i) The defect arrivals or defects reported during testing phase by time interval (weeks) - These are raw number of arrivals and not each one is a valid defect.

  (ii) The pattern of valid defect arrivals - It is true defect pattern where reported problems is determined as defect.

  (iii) The pattern of defect backlog overtime - Its difficult for organizations to investigate and fix all reported problems urgently. This metric serves as a workload statement and a quality statement. Large number of defect backlog at the end of SDLC and many remaining fixes yet to be integrated into the system, will affect stability and ultimately quality of system. Regression testing is required to ensure that targeted product quality levels are reached.

6.3.3 Phase-Based Defect Removal Pattern

- The phase-based defect removal pattern reflects overall defect removal ability of development process. It is an extension of test defect density metric. It needs defect tracking at all phases of the SDLC like design reviews, code inspections, formal verification before tracking testing defects.
- Conduction of formal reviews or functional verification can maximize process' defect removal capability. It will minimize the error injection at front end since large number of programming defects is related to design problems.
- Metrics called as, **"inspection coverage"** and **"inspection effort"** for in-process quality management is also used along with defect rate metric. Setup of "model values" and "control boundaries" for various in-process quality indicators (e.g. review manpower rate, review work hours, review design hours, review coverage rate, defect rate etc.) is also done by many organizations.
- Fig. 6.3.3 shows the defect removal pattern of development project A (front-end loaded) and development project B (heavily testing-dependent) to remove defects. Various phases of defect removal are I0 (High-level design review), I1 (Low-level design review), I2 (Code Inspection), UT (Unit Test), CT (Component Test), and ST (System Test). Conclusion is field quality of project A outperformed project B significantly.

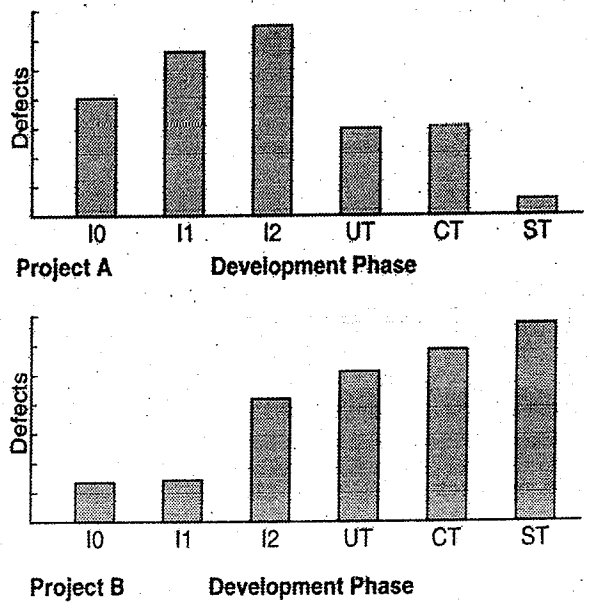


Fig 6.3.3 : Defect Removal by Phase for Two Products

6.3.4 Defect Removal Effectiveness (DRE)

- DRE is defined as follows

$$DRE = \frac{\text{Defects removed during a development phase}}{\text{Defects latent in the product}} \times 100$$

- Latent defects can be calculated by adding defects removed during the phase and defects found later. DRE can be calculated for complete development process, for front end before code integration (known as **early defect removal**), and for each phase (known as **phase effectiveness**).
- High value of DRE means more effective development process and very few defects escape to next phase or to field. DRE is very important metric of the defect removal model for software development.
- Fig. 6.3.4 shows the DRE for specific phases in a software project. The weakest phases are UT (unit test), I2 (code inspections), and CT (component test). By looking at DRE metric values, action plans are established and deployed to improve the effectiveness of these phases.

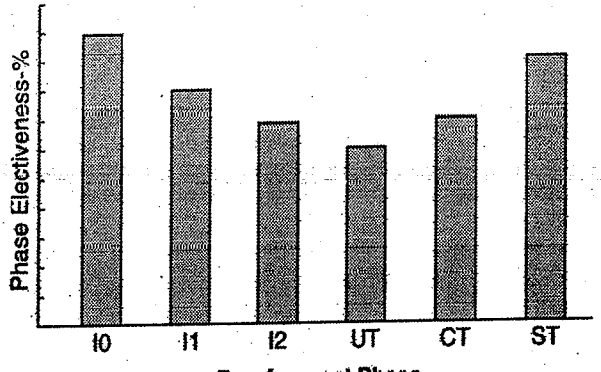


Fig 6.3.4 : Phase Effectiveness of a Software Project

Syllabus Topic : Software Maintenance

6.4 Software Maintenance

- Once software is developed and released, it enters in to the maintenance phase of SDLC. By default there exists two metrics such as - defect arrivals by time interval and customer problem calls (which may or may not be defects) by time interval.

- Development process decides the number of defect or problem arrivals. Nothing is done to the product quality in maintenance phase hence for mentioned two metrics gives no reflection of quality.
- Fixing defects as soon as possible and with excellent fix quality will not improve the defect rate of the product but can improve customer satisfaction to a large extent. Following section explains four software maintenance metrics.

6.4.1 Fix Backlog and Backlog Management Index

- Fix backlog is a workload statement for software maintenance that defines relation between rate of defect arrivals and rate at which fixes for reported problems become available.
- Fix backlog gives count of reported problems that remain at the end of each month/week. Trend chart representation helps to manage maintenance process.
- Backlog Management Index (BMI) is used to manage the backlog of open, unresolved, problems. It is the ratio of number of closed, or solved, problems to number of problem arrivals during the month.

$$BMI = \frac{\text{Number of problems closed during the month}}{\text{Number of problem arrivals during the month}} \times 100\%$$

- BMI greater than 100 means the backlog is reduced and If less than 100, then backlog is increased. Using data points and control charts backlog management capability of the maintenance process can be calculated. If BMI value goes out of control then use more investigation and analysis along with trend charts of defect arrivals, defects fixed (closed), and the number of problems in the backlog.
- Fig. 6.4.1 shows a trend chart of month against numbers of opened and closed problems. It also shows a pseudo-control chart for the BMI. The latest release of the product was available to customers in the month for the first data points on the two charts.
- This explains the rise and fall of the problem arrivals and closures. The mean BMI was 102.9%, indicating that the capability of the fix process was functioning normally. All BMI values were within the upper (UCL) and lower (LCL) control limits the backlog management process was in control.

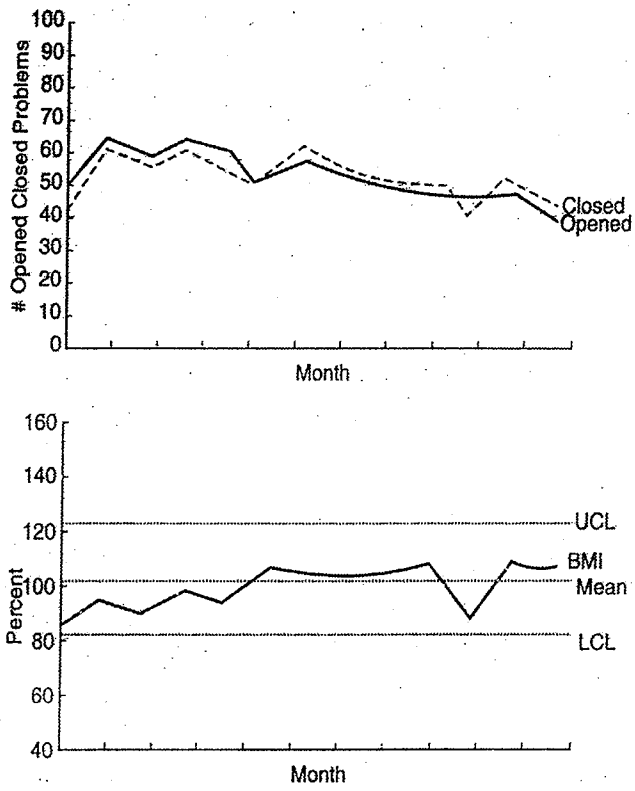


Fig. 6.4.1 : Opened Problems, Closed Problems and Backlog Management Index by Month

6.4.2 Fix Response Time and Fix Responsiveness

- The fixes should be available for reported defects within given time limit. Many organizations are setting guidelines for within time fixes based on severity of problems.
- In critical situations when customers' businesses are at risk due to defects in product, developers/maintenance teams work hard to fix the problems. For less severe defects the required fix time is more relaxed.
- The fix response time metric is calculated for all problems with all severity level as : **"Mean time of all problems from open to closed."**
- For extreme data points values (observed in case of less severe defects), median is used instead of mean. Smaller fix response time leads to customer satisfaction.
- In the metric fix responsiveness important element is customer expectations. It is deals with agreed-to fix time, and the ability to meet commitment made to customer. Automobile dealers (motor service department) uses practice of fix responsiveness process that focus on customer satisfaction. Same process is followed by Hewlett-Packard (HP) and IBM.

6.4.3 Percent Delinquent Fixes

- Delinquent fix are those for which the turnaround time greatly exceeds the required response time. It is more sensitive metric than mean (or median) response time metric.

$$\text{Percent delinquent fixes} = \frac{\text{Number of fixes that exceeded the response time criteria by severity level}}{\text{Number of fixes delivered in a specified time}} \times 100\%$$

- This metric is not related to real-time delinquent management (where problems are still open), instead it focuses on closed problems only.
- Assuming that the time is 1 week, and the percent delinquent of problems in the active backlog be used. Active backlog means total number of problems to be processed for the week i.e. total workload.
- It contains all opened problems for the week, which is the sum of existing backlog at the beginning of week and new problem arrivals during same week. At the end of week the number of delinquent problems are checked. Fig. 6.4.2 shows the real-time delivery index.

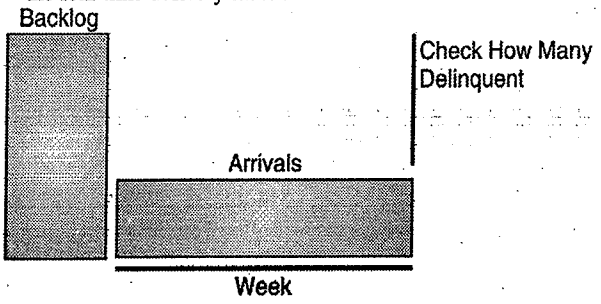


Fig. 6.4.2 : Real-Time Delinquency Index

6.4.4 Fix Quality

- This metric is an important quality metric for the maintenance phase that refers to the number of defective fixes. From the customer's perspective, getting encounter functional defects is not acceptable. A defective fix is either did not fix the reported problem, or it fix the original problem but inject a new defect. For critical software applications, defective fixes loses customer satisfaction.
- A defective fix can be recorded as the month it was discovered (customer measure) or the month the fix was delivered (process measure).
- The difference between above two events/dates is termed as latent period of defective fix. Information of latency data and number of customers who were affected by the defective fix is very important and need to be keep. Longer latency affects more customers because there is more time for customers to apply defective fix to their application.

Syllabus Topic : Ishikawa's 7 Basic Tools, Checklist, Pareto Diagrams, Histogram, Run Charts, Scatter Diagrams, Control Chart and Cause effect Diagram

6.5 Ishikawa's 7 Basic Tools

It consists a set of general tools useful for planning or controlling project quality. It was introduced by Kaoru Ishikawa, a Japanese professor of Faculty of Engineering. The seven tools are :

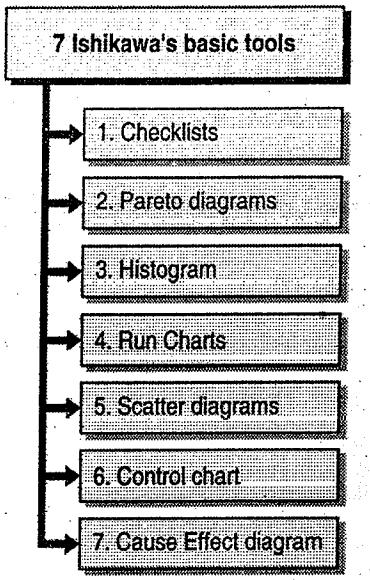


Fig. 6.5.1 : 7 Ishikawa's Basic tools

These seven tools are illustrated in following section.

6.5.1 Checklists

- The purpose of check sheet is for collecting and organizing measured or counted data.

- Data collected can be used as input data for other quality tools
  - A check sheet is a structured sheet, prepared form for collecting and analyzing data.
  - It is a a generic tool that can be adapted for a wide variety of purposes.
  - The check sheet prepared based on the location where the data is created.
  - To collect check sheet data on the frequency, location, or even cause of problems or defects that occur.
  - The check sheet is divided into a number of different regions, and data is then marked into the different regions using a mark to check to indicate something.

Benefits

1. Collect data in a systematic and organized manner.
2. To determine source of problem.
3. To facilitate classification of data (stratification).

- Another type of checklist is the common error list, which is part of the stage kickoffs of the Defect Prevention Process (DPP).
- DPP involves three key steps :
  (1) Analysis of defects to trace the root causes,
  (2) Action teams to implement suggested actions,
  (3) Stage kickoff meetings as the major feedback mechanism.

Motor Assembly Check Sheet

Name of Data Recorder: Lester B. Rapp
Location: Rochester, New York
Data Collecion Dates: 1/17 - 1/23

Defect Types/ Event Occurrence	Dates							TOTAL
	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	
Supplied parts rusted		|||||	|||||	||||	||			20
Misaligned weld			|||		||		5
Improper test procedure								0
Wrong part issued								3
Film on parts								0
Voids in casting				||||	||			6
Incorrect dimensions						||		2
Adhesive failure								0
Masking insufficient								1
Spray failure			|||||				5
TOTAL		10	13	10	5	4		

Fig. 6.5.2 : Sample of Check sheet

6.5.2 Pareto Diagrams

The purpose of Pareto diagrams/charts is to prioritize problems

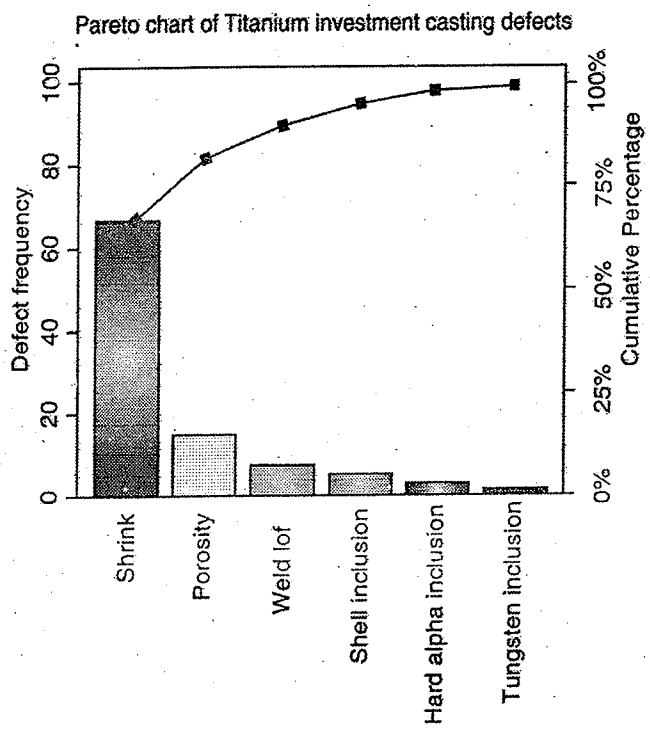


Fig. 6.5.3 : Sample of Pareto chart

How is it done?

- Create a preliminary list of problem classifications.
- Tally the occurrences in each problem classification.
- Arrange each classification in order from highest to lowest
- Construct the bar chart

Benefits

1. Pareto analysis helps graphically display results so the significant few problems emerge from the general background.
2. It tells you what to work on first.

6.5.3 Histogram

- Histograms were first introduced by Karl Pearson. A histogram is a graph that shows how often a value, or range of values, occurs within a given time period.
- It provides a visual summary of large amounts of variable data. It is the easiest way to evaluate the distribution of data.
- To determine the spread or variation of a set of data points in a graphical form.

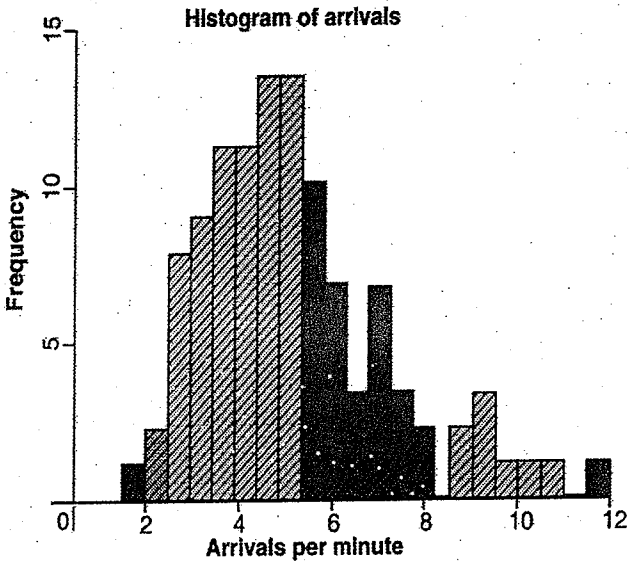


Fig. 6.5.4 : Sample of Histogram

Creating a Histogram

- Collect data, 50-100 data point.
- Determine the range of the data.
- Calculate the size of the class interval.
- Divide data points into classes Determine the class boundary.
- Count of data points in each class.
- Draw the histogram.

The patterns of the histogram are described as

1. Symmetric
2. Skewed left
3. Skewed right
4. Unimodal
5. Bi-modal
6. Multimodal

Benefits

1. Allows you to understand at a glance the variation that exists in a process.
2. The shape of the histogram will show process behavior.
3. Often, it will tell you to dig deeper for otherwise unseen causes of variation.

4. The shape and size of the dispersion will help identify otherwise hidden sources of variation.
5. Used to determine the capability of a process.
6. Starting point for the improvement process.

6.5.4 Run Charts

- Stratification is a technique that separates data gathered from a variety of sources so that patterns can be seen, some lists replace "stratification" with "flowchart" or "run chart".
- Stratification is a technique used in combination with other data analysis tools. When data from a variety of sources or categories have been lumped together, the meaning of the data can be impossible to see. This technique separates the data so that patterns can be seen.

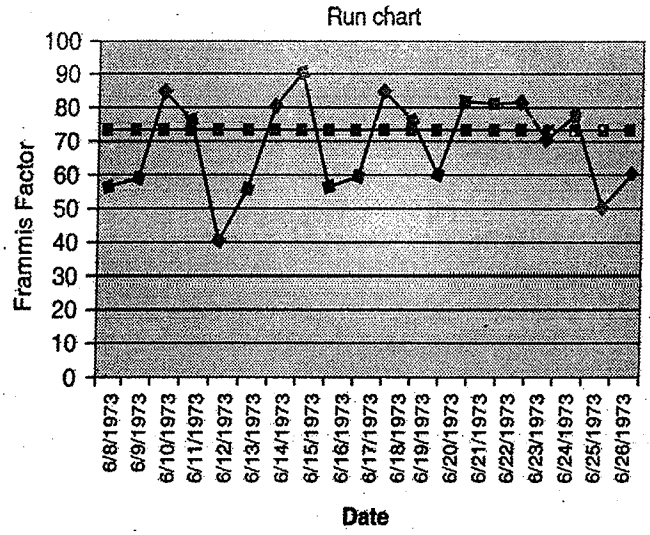


Fig. 6.5.5 : Sample of run chart

Run Chart

- A run chart also known as a **run-sequence plot** is a graph that displays observed data in a time sequence.
- A run chart helps you to analyze the following :
  1. Trends in the process ; i.e. whether the process is moving upward or downward.
  2. Trends in the output of the manufacturing process.
  3. If the process has any cycle or any shift.
  4. If the process has any non-random pattern in behavior over a period of time.

Creating a Run Chart

1. Gathering Data
2. Organizing Data
3. Charting Data
4. Interpreting Data

6.5.5 Scatter Diagrams

- A scatter diagram displays the relationship of two interval variables. Scatter diagrams are also called **scatter graphs, scatter charts, scatter plots, and even scatter-grams.** Scatter diagrams are often used to help understand how variables are related and also to identify root cause. Compared to other tools, the scatter diagram is more difficult to apply.
- To identify the correlations that might exist between a quality characteristic and a factor that might be driving it.
- A scatter diagram shows the correlation between two variables in a process.
  - These variables could be a Critical To Quality (CTQ) characteristic and a factor affecting it two factors affecting a CTQ or two related quality characteristics.
- Dots representing data points are *scattered* on the diagram.
- The extent to which the dots cluster together in a line across the diagram shows the strength with which the two factors are related.

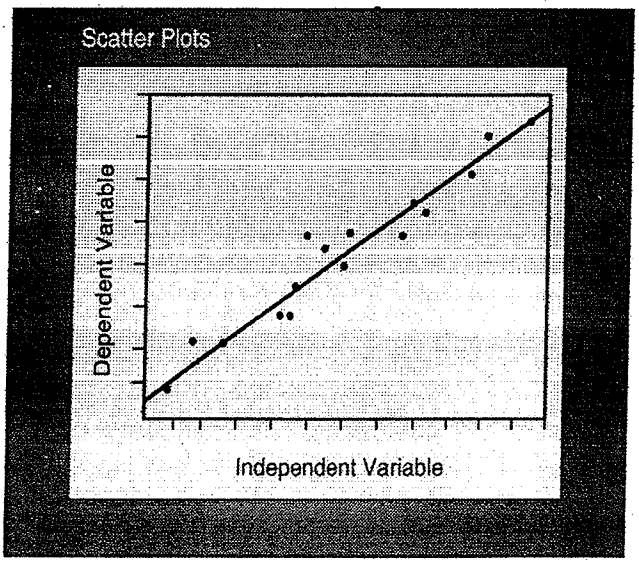


Fig. 6.5.6 : Sample of Scatter Diagram

Constructing a Scatter Diagram

- First, collect two pieces of data and create a summary table of the data.
- Draw a diagram labeling the horizontal and vertical axes.
- It is common that the "cause" variable/independent variable be labeled on the X axis and the "effect" variable/dependent variable be labeled on the Y axis.
- Plot the data pairs on the diagram. You can then draw a trend line to study the relationship between the variables.
- At last interpret the scatter diagram for direction and strength.

Benefits

1. Helps identify and test probable causes.
2. By knowing which elements of your process are related and how they are related, you will know what to control or what to vary to affect a quality characteristic.

6.5.6 Control Chart

- You can use a process control chart to track the values of a process over time. They're also known as Shewhart charts (after Walter A. Shewhart) or process-behavior charts.
- It consists of a central line represents the average or mean, and two parallel lines (above and below that center line) that represent the upper and lower control limits.values of the parameter of interest plotted on the chart which represent the state of a process.

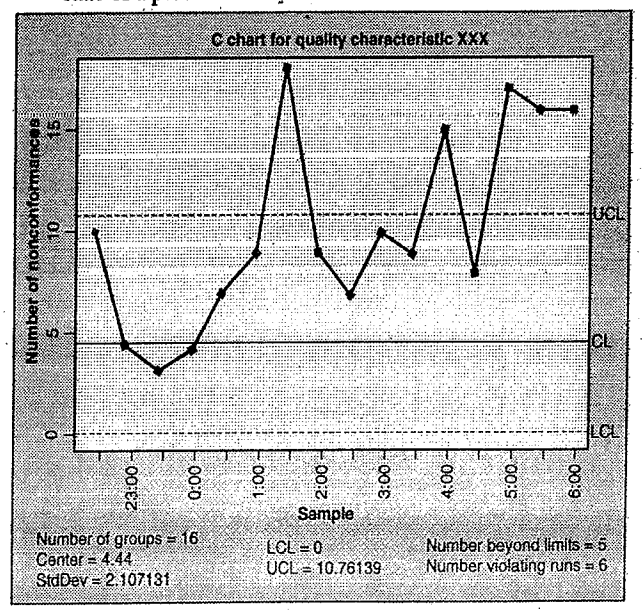


Fig. 6.5.7 : Sample of Control chart

The vertical dimension of the chart usually represents a process value or measurement, and the horizontal dimension usually represents the average or mean, and the dotted lines represent the control limits.

How is it done?

- The data must have a normal distribution (bell curve).
- Have 20 or more data points. Fifteen is the absolute minimum.
- List the data points in time order. Determine the range between each of the consecutive data points.
- Find the mean or average of the data point values.
- Calculate the control limits (three standard deviations).
- Set up the scales for your control chart.
- Draw a solid line representing the data mean.
- Draw the upper and lower control limits.
- Plot the data points in time sequence.

Benefits

1. Predict process out of control and out of specification limits
2. Distinguish between specific, identifiable causes of variation
3. Can be used for statistical process control

6.5.7 Cause Effect Diagram

- Ishikawa diagrams are named after their inventor, Kaoru Ishikawa. It looks like a child's drawing of a fish skeleton which is used to show the cause of some effect.
- They are also called **fishbone charts**, after their appearance, or cause and effect diagrams after their function.
- Their function is to identify the factors that are causing an undesired effect (e.g., defects) for improvement action, or to identify the factors needed to bring about a desired result (e.g., a winning proposal). The factors are identified by people familiar with the process involved.

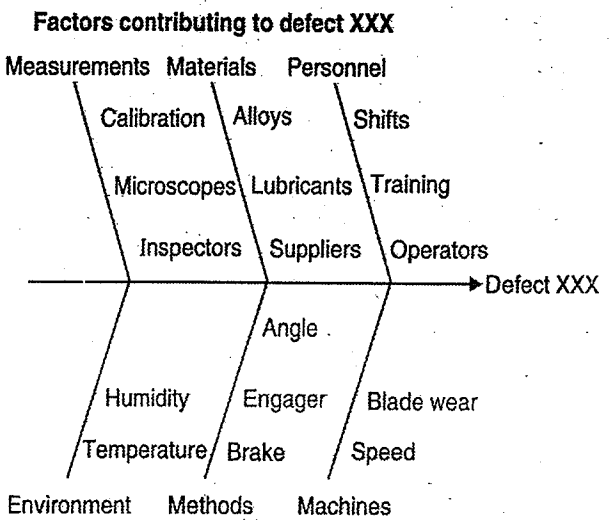


Fig. 6.5.8 : Fishbone diagram

Table 6.5.1 : Categories of causes

The 8 Ms (used in manufacturing)	The 8 Ps (used in service industry)	The 4 Ss (used in service industry)
Machine (technology)	Product=Service Place	Surroundings
Method (process)	Process	Suppliers
Material (Includes Raw Material, Consumables and Information.)	Productivity & Quality	Systems
Man Power (physical work)/Mind Power (brain work)	People (key person)	Skills
Measurement (Inspection)	Price	
Milieu/Mother Nature (Environment)	Promotion/Entertainment	
Management/Money Power	Physical Evidence	
Maintenance	Place	

- Categories can be subdivided, if useful, and the identification of significant factors is often a prelude to the statistical design of experiments.

Benefits

1. Breaks problems down into bite-size pieces to find root cause.
2. Fosters team work.
3. Common understanding of factors causing the problem.
4. Road map to verify picture of the process.
5. Follows brainstorming relationship.

6.5.7(A) Constructing a Cause and Effect Diagram using 4 Steps

- It is most important tasks in building a cause and effect diagram.
- While defining your problem or event, your problem statement may also contain information about the location and time of the event.
- On the cause and effect diagram the problem is visually represented by drawing a horizontal line with a box enclosing the description of the problem on the tip of the arrow.

Fig. 6.5.9 : Identify the key causes of the problem or event.

- Using brainstorming techniques find the primary causes of the problem.
- Categorized these causes are categorized under people, equipment, materials, external factors, etc. Such as 8M's, 8P's and 4S's.
- The image below shows how to visually depict these key causes on the cause and effect diagram.

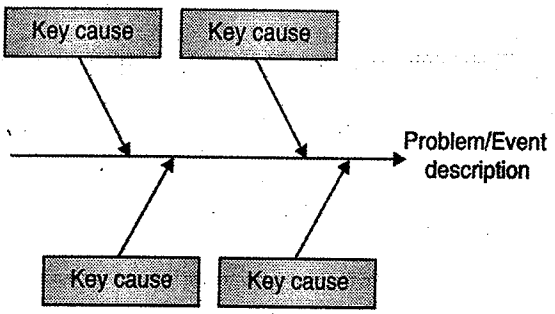


Fig. 6.5.10 : Identify the reasons behind the key causes

- The goal in this step is to brainstorm as many causes for each of the key causes.

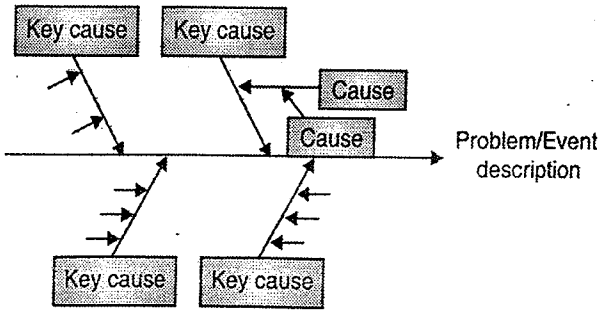


Fig. 6.5.11 : Identify the most likely causes.

- At the end of step three, your team should have a good overview of the possible causes for the problem or event; if there are areas in the chart where possible causes are few, see if your team can dig deeper to find more potential causes.

Syllabus Topic : Defect Removal Effectiveness and Process Maturity Level

6.6 Defect Removal Effectiveness and Process Maturity Level

- One of the expensive activity in any software project that has greater impact on project schedule is Defect removal. Effective defect removal can reduces development cycle time thus produces a good product.
- It is important for all development organizations to measure the effectiveness of their defect removal processes. It talks about ability of project or organization to find and fix defects, so that they do not go to the customer.
- DRE is termed as a process metric. Defect Removal Effectiveness (or efficiency), DRE, is calculated as follows :

$$DRE = \frac{\text{Defects removed during a development phase}}{\text{Defects latent in the product at that phase}} \times 100\%$$

- At any point of time, the latent defects in the software are unknown. These defects can be approximated by adding the number of defects removed during the phase to the number of defects found later (but that existed during that phase).

☞ Definitions related to DRE

1. Error detection efficiency

$$\text{Error detection efficiency} = \frac{\text{Errors found by an inspection}}{\text{Total errors in the proudct before inspection}} \times 100$$

2. Removal efficiency

$$\text{Removal efficiency} = \frac{\text{Defects found by removal operation}}{\text{Defects present at removal operation}} \times 100\%$$

$$= \frac{\text{Defect found}}{\text{Defects found} + \frac{\text{Defects not found}}{\text{(found later)}}} \times 100\%$$

3. Early detection percentage

$$\text{Early detection percentage} = \frac{\text{Number of major inspection errors}}{\text{Total number of errors}} \times 100\%$$

4. Effectiveness measure

$$E = \frac{N}{N+S} \times 100\%$$

where;

E = Effectiveness of activity (development phase).

N = Number of faults (defects) found by activity (phase).

S = Number of faults (defects) found by subsequent activities (phases).

☞ Metrics for DRE

1. Total defect containment effectiveness (TDCE)

$$TDCE = \frac{\text{Number of pre-release defects}}{\text{Number of pre-release defects} + \text{Number of post-release defects}}$$

2. Phase containment effectiveness (PCE)

$$PCE = \frac{\text{Number of phase i errors}}{\text{Number of phase i errors} + \text{number of phase i defects}}$$

where;

- phase i errors = problems found during that development phase in which they were introduced,
- phase i defects = problems found later than the development phase in which they were introduced.

6.6.1 Defect Injection and Defect Removal Related Activities

- Table 6.6.1 lists some of the activities in which defects can be injected or removed for a development process. For the development phases before testing, the defect injection is in the development activities only.
- Reviews or inspections at end-of-phase activities are the important for defect removal. For the testing phases, defect removal is through the testing itself.
- Defects and bad fixes are possible; if the problems found by testing are fixed incorrectly, or after inspection steps also.

Table 6.6.1 : List of activities related to defect injection and removal

Development Phase	Defect Injection	Defect Removal
Requirements	Requirements-gathering process and the development of programming functional specifications.	Requirement analysis and review.
High-level design	Design work.	High-level design
Low-level design	Design work.	Low-level design inspections.
Code implementation	Coding.	Code inspections.
Integration/Build	Integration and build	Build
Unit test	Bad fixes.	Testing itself.
Component test	Bad fixes.	Testing itself.
System test	Bad fixes.	Testing itself.

– Fig. 6.6.1 illustrates the detailed mechanics of defect injection and removal at each step of the development process.

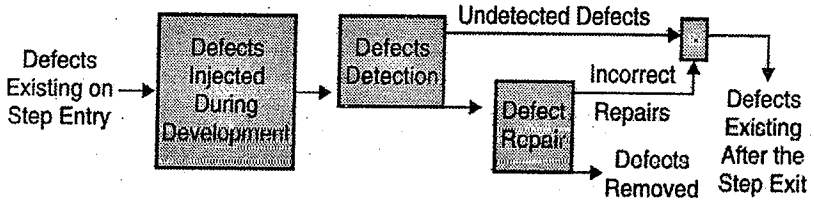


Fig. 6.6.1 : Defect Injection and Removal During One Process Step

From the Fig. 6.6.1, the definition of DRE for each development step can be defined as :

$$\frac{\text{Defects removed (at the step)}}{\text{Defects existing on step entry + Defects injected during development (of the step)}} \times 100\%$$

Ex. 6.6.1

Consider Fig. P. 6.6.1 that represents defect data in development phase where the defects are found (and then removed) and the phases in which the defects are injected termed as defect origin. For each defect found, its origin should be decided by the inspection group (for inspection defects) or by tester and developer (for testing defects).

Where Found \ Defect Origin	Requirements	High Level Design	Low Level Design	Code	Unit Test	Component Test	System Test	Field	Total
RQ	–			–					
I0	49	681							730
I1	6	42	681						729
I2	12	28	114	941					1095
UT	21	43	43	223	2				332
CT	20	41	61	261	–	4			387
ST	6	8	24	72	–	–	1		111
Field	8	16	16	40	–	–	–	1	81
Total	122	859	939	1537	2	4	1	1	3465

Fig. P. 6.6.1 : Defect Data Cross-Tabulated by Where Found and Defect Origin

1. High-Level Design Inspection Effectiveness; IE (I0)

- Defects removed at row entry I0 : 730
- Defects existing on step entry or those escapes from requirements phase) : 122
- Defects injected in current phase (high level design) : 859

$$IE(I0) = \frac{730}{122+859} \times 100\% = 74\%$$

2. Low-Level Design Inspection Effectiveness; IE (I1)

- Defects removed at I1 : 729
- Defects existing on step entry (escapes from requirements phase and I0) :

$$122 + 859 - 730 = 251$$

- Defects injected in current phase: 939

$$IE(I1) = \frac{729}{251+939} \times 100\% = 61\%$$

3. Code Inspection Effectiveness; IE (I2)

- Defects removed at I1 : 1095
- Defects existing on step entry (escapes from requirements phase, I0 and I1) :

$$122 + 859 + 939 - 730 - 729 = 461$$

- Defects injected in current phase : 1537

$$IE(I2) = \frac{1095}{461+1537} \times 100\% = 55\%$$

4. Unit Test Effectiveness; TE (UT)

Defects removed at I1 : 332

Defects existing on step entry (escapes from all previous phases) :

$$122 + 859 + 939 + 1537 - 730 - 729 - 1095 = 903$$

Defects injected in current phase (bad fixes): 2

$$TE(UT) = \frac{332}{903+2} \times 100\% = \frac{332}{905} \times 100\% = 37\%$$

- For the testing phases, the defect injection (bad fixes) is usually a small number. In such cases, effectiveness can be calculated with the help of some other method.

Effectiveness

$$= \frac{\text{Defects removed at current phase}}{\text{Defects removed at current phase + Defects removed at subsequent phases}} \times 100\%$$

$$TE(UT) = \frac{332}{322+387+111+81} \times 100\%$$

$$= \frac{332}{901} \times 100\% = 36\%$$

5. Component Test Effectiveness; TE (CT)

$$TE(CT) = \frac{387}{387+111+81} \times 100\% = 67\%$$

6. System Test Effectiveness; TE (ST)

$$TE(ST) = \frac{111}{111+81} \times 100\% = 58\%$$

7. Overall Inspection Effectiveness; IE

$$IE = \frac{730+729+1095}{122+859+939+1537} \times 100\%$$

$$= \frac{2554}{3457} \times 100\% = 74\%$$

OR

$$IE = \frac{\text{Defects removed by inspection}}{\text{All defects}} \times 100\%$$

$$= \frac{730+729+1095}{3465} \times 100\% = 74\%$$

8. Overall Test Effectiveness; TE

$$TE = \frac{332+387+111}{332+387+111+81} \times 100\% = 91\%$$

9. Overall Defect Removal Effectiveness of the Process; DRE

$$DRE = \left(1 - \frac{81}{3465}\right) \times 100\% = 97.7\%$$

- Phase containment effectiveness means the ability of the phase inspection to remove defects introduced during a particular phase, whereas DRE means the overall ability of the phase inspection to remove defects that were present at that time. DRE includes the defects introduced at that particular phase as well as defects that escaped from previous phases.
- Hence, PCE values will be higher than the DRE values based on the same data. The PCEi values of above example are as follows.

I0 = 681/(42 + 28 + 43 + 41 + 8 + 16) = 681/859 = 79%

I1 : 681/939 = 73%

I2 : 941/1537 = 61%

UT : 2/2 = 100%

CT : 4/4 = 100 %

ST : 1/1 = 100%

Table P.6.6.1 : Summary of metric values

DRE (in %)		PCE (in %)	
I0	74	I0	79
I1	61	I1	73
I2	55	I2	61
UT	36	UT	100
CT	67	CT	100
ST	58	ST	100

☞ **DRE and Process Maturity Level**

– Following are the DRE estimates for organizations at different levels of the development process capability maturity model (CMM) :

Level 1 : 85%

Level 2 : 89%

Level 3 : 91%

Level 4 : 93%

Level 5 : 95%

– These values can be used as comparison baselines for organizations to evaluate their relative capability with regard DRE.

Review Questions

Q. Explain total project quality management. **(Section 6.1)**

Q. What are the TQM frameworks? List out key benefits of TQM. **(Sections 6.1.3 and 6.1.4)**

Q. Describe key elements and other important elements of a TQM system. **(Section 6.1.1)**

Q. Explain with example in-process quality metric - Defect Arrival Pattern During Machine Testing. (Section 6.3.2)

Q. Explain with example in-process quality metric - Defect Removal Effectiveness. **(Section 6.3.4)**

Q. Explain with example product quality metric - The Defect Density Metric **(Section 6.2.1)**

Q. Define following metric in brief -
(i) Customer Problems Metric (Section 6.2.2)
(ii) Customer Satisfaction Metrics **(Section 6.2.3)**

Q. Explain with example software maintenance metric - Fix Backlog and Backlog Management Index **(Section 6.4.1)**

Q. Explain with example software maintenance metric - Percent Delinquent Fixes. (Section 6.4.3)

Q. Explain seven basic tools of software quality? **(Section 6.5)**

Q. Explain following quality tools -
(I) Cause and effect diagram **(Section 6.5.7)**
(II) Histogram **(Section 6.5.3)**

Q. Give examples of following quality tools and explain it in a brief way -
(I) Pareto chart **(Section 6.5.2)**
(II) Control chart **(Section 6.5.6)**

Q. Write a note on following Ishikawa Tools of quality.
(I) Check sheet **(Section 6.5.1)**
(II) Scatter diagram **(Section 6.5.5)**

Q. Explain Defect Removal Effectiveness. **(Section 6.6)**

Q. List the activities related to defect injection and removal. **(Section 6.6.1)**

□□□

Note