SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

School of Computing

Department of Computer Science and Engineering

and

Department of Information Technology

SCS1608 - Software Quality Assurance and Testing

UNIT I

Course Outcomes:

1. Comprehend the concepts of Software Quality Standards.
2. Model a Software Quality Assurance Framework & standards for analyzing quality metrics.
3. Develop product and process quality metrics using different principles.
4. Construct test cases using variety test strategies for any given applications.
5. Perceive knowledge on different Software testing tools.
6. Design an appropriate testing methodology & tool for any software requirements.

INTRODUCTION

Software Quality

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally develop software

Three (3) important points to remember on software quality

1. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
2. Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
3. There is a set of implicit requirements that often goes unmentioned (e.g., the desire for good maintainability). If software conforms to its explicit requirements, but fails to meet implicit requirements.

Quality Characteristics

Is any property or element that can be used to define the nature of a product. Each characteristic can be physical or chemical properties such as size, weight, volume, color or composition.

Software Quality

1. Is achieved through a disciplined approach - called software engineering SE
2. Can be defined, described, and measured
3. Can be assessed before any code has been written

4 Cannot be tested into a product

Software quality challenges

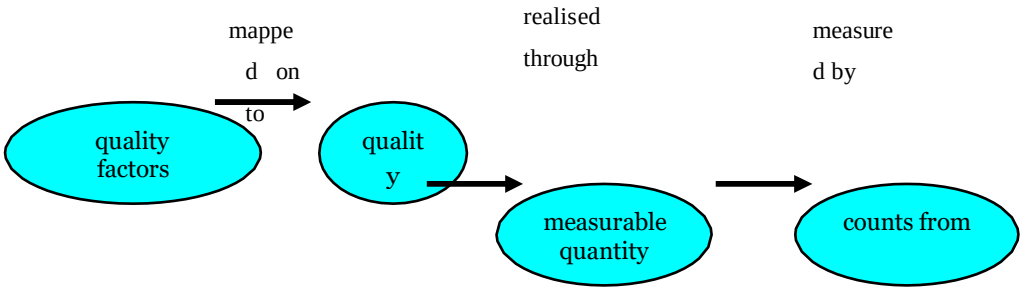
1. Defining it
2. Describing it (qualitatively)
3. Measuring it (quantitatively)
4. Achieving it (technically)

Designing software is a *creative* task, and like most such tasks, success is more likely if

the designer follows what might be termed a set of *rules of form.*The rules of form also provide some way of assessing the *quality* of the eventual product, and possibly of the processes that led to it.

REALISING QUALITY

A set of abstract quality factors (‘the *ilities*’) has been defined. These cannot be measured directly but do relate to the ultimate goal.



Software Quality Factors (by McCall)

1) **Product Revision (changing it)**

- *Flexibility (can I change it?)*

  The effort required to modify an operational program. Change and enhancement of the system should be easily implementable.

- *Maintainability (can I fix it?)*

  The effort required to locate and fix an error in a program. The system should be easy to keep up for its intended use. Changes for improving operational efficiency should be easy to implement. Failed operations should be easy to restore to satisfactory condition.

- *Testability (can I test it?)*

  The effort required to test a program to ensure that it performs its intended function . The ability of the system to produce quality product units should be easily testable. Useful messages should be generated for testing and debugging purposes.

2) Product Transition (modifying it to work in a different environment)

Interoperability (Will I be able to interface it with another system?)

The effort required to couple one system to another.

- *Portability (Will I be able to use it on another machine?)*

  The effort required to transfer the program from one hardware and/or software system environment to another. The system should be portable among people and among machines. Attainment of theother quality characteristics greatly facilitates portability.

- *Reusability (Will I be able to reuse some of the software?)*

The extent to which a program (or part of a program) can be reused in other applications-related to the packaging and scope of the functions that the program performs.

3) Product Operations (using it)

- *Correctness (Does it do what I want?)*
  The extent to which a program satisfies its specification and fulfills the customer's mission objectives. The extent to which software is free from design defects and from coding defects.; that is fault-free.

- *Reliability (Does it do it accurately all of the time?)*
  The extent to which a program can be expected to perform its intended function with required precisions under stated conditions for a stated period of time.

- *Efficiency (Will it run on my hardware as well as it can?)*
  The extent to which a software performs its function.with a minimum consumption of computing resources. It should not use any hardware components or peripheral equipment unnecessarily.

- *Integrity (Is it secure?)*
  The extent to which access to software or data by unauthorized persons can be controlled.

- *Usability (Is it designed for the use?)*
  The effort required to learn, operate, prepare input, and interpret output of a program.

Software Quality Assurance (SQA)

Software quality assurance is a planned effort to ensure that a software product fulfills these criteria and has additional attributes specific to the project, e.g., portability, efficiency, reusability, and flexibility. It is the collection of activities and functions used to monitor and control a software project so that specific objectives are achieved with

the desired level of confidence. It is not the sole responsibility of the software quality assurance group but is determined by the consensus of the project manager, project leader, project personnel, and users.

A formal definition of software quality assurance is that is 'the systematic activities providing evidence of the fitness for use of the total software product." Software quality assurance is achieved through the use of established guidelines.for quality control to ensure the Integrity and prolonged life of software. The relationships between quality assurance, quality control, the auditing function, and software testing are often confused.

Quality assurance is the set of support activities needed to provide adequate confidence that processes are established and continuously improved in order to products that meet specifications and are fit for use. Quality control is the process by which product quality is compared with applicable standards and the action taken when nonconformance is detected. Auditing is the inspection/ assessment activity that verifies compliance with plans, policies, and procedures.

SQAActivities

SQA is ensured through a Quality Management System (QMS), QMS is made of several components; it is a system integrated in the bigger system of software development, which comprises project, process and product management systems.

The Software Engineering Institute (SEI) recommends a set of activities, which, when implemented effectively, assures the designed quality. These activities include:

- Quality assurance planning
- Data gathering on key quality defining parameters
- Data analysis and reporting
- Quality control mechanisms

The first and foremost requirement in SQA is that it is a separate group responsible for quality in the organization. They set the goals, standards and mechanisms (systems) for SQA. The role of the SQA group is to assist the software development team in managing

the quality requirements of the software. Every software has certain quality goals specified by the customer. These quality goals are to be achieved by the development team by introducing a set of activities or ensuring the delivery of quality to the customer.

SQA activities operate on the normal activities of quality management. These activities play the role of monitoring, tracking, evaluations, auditing and reviews to ensure that the quality policy of the organization is implemented. These activities are independently carried out, and feedback is given to the development team. The responsibility of delivering the required quality to the customer rests with the development team. The development team has an obligation to implement quality policy in terms of goals, objectives, procedures, checks and controls, documentation and feedback to management. For example, the quality policy stipulates preparation of a test plan for stages for development as well as at the end of the development process. SQA has a variety of tools to implement the policy.

They are

- Auditing
- Inspection

Verify compliance with those norms and practices specified in QA policy; deviations are set right. Ensure that deviations are documented and reported and put into the QA database for guidance. Design and architecture is reviewed to ensure that standards are met and customer quality is assured. Implement change management. Collect data on various observations in the process of auditing, inspection and reviews to build QA database and to improve various standards.

Software Defects

The causes for not meeting the quality commonly are

- Imprecise requirement and software specifications
- Lack of understanding of customer requirements
- International deviations
- Violation of standards (Design, Programming)
- Erroneous data representation

- Improper interface
- Faulty logic in rules and processes
- Erroneous testing
- Incomplete and defective documentation
- H&S platforms not coping up to required standards
- Lack of domain knowledge

Statistical analysis of errors or defects helps to focus and concentrate on probable areas where SQA efforts are necessary.

SQA is also concerned with two other aspects namely, software reliability and software safety. Software reliability is defined as the probability of failure free operation of a computer program in a specified environment for a specified time.

The nature of failure may be such that one error may require only a few repair, and other may need hours. SQA collects data on these failures and examines why these failures could not be prevented through earlier SQA activities.

A simple measure of reliability is Mean Time between Failure (MTBF).
MTBF = MTTF + MTTR
Where MTTF is Mean Time to Failure and MTTR is Mean Time to Repair.

Software safety deals with identification and assessment of potential hazards of software failing, and its impact on the system or in the environment in which it operates. Software safety needs are more crucial in process control systems, health care systems, defense and so on. SQA activities concentrate on such areas of software where failure affects the customer system adversely. In short, SQA efforts assure software quality, reliability, availability and safety.

Software Quality Assurance Components

The SQA Components that are used by the Software Quality Assurance System can be classified into

six different categories; each of which is necessary to guarantee maximum quality and to ensure the compliance with the standards and procedures.

"The environment in which software development and maintenance is undertaken directly influences the SQA components. Alongside this various errors will also affect the SQA components; therefore it is usually necessary to include all of the components."

casd.csie.ncku.edu.tw/sq/ch04.ppt

The six different components are broken down into the following categories

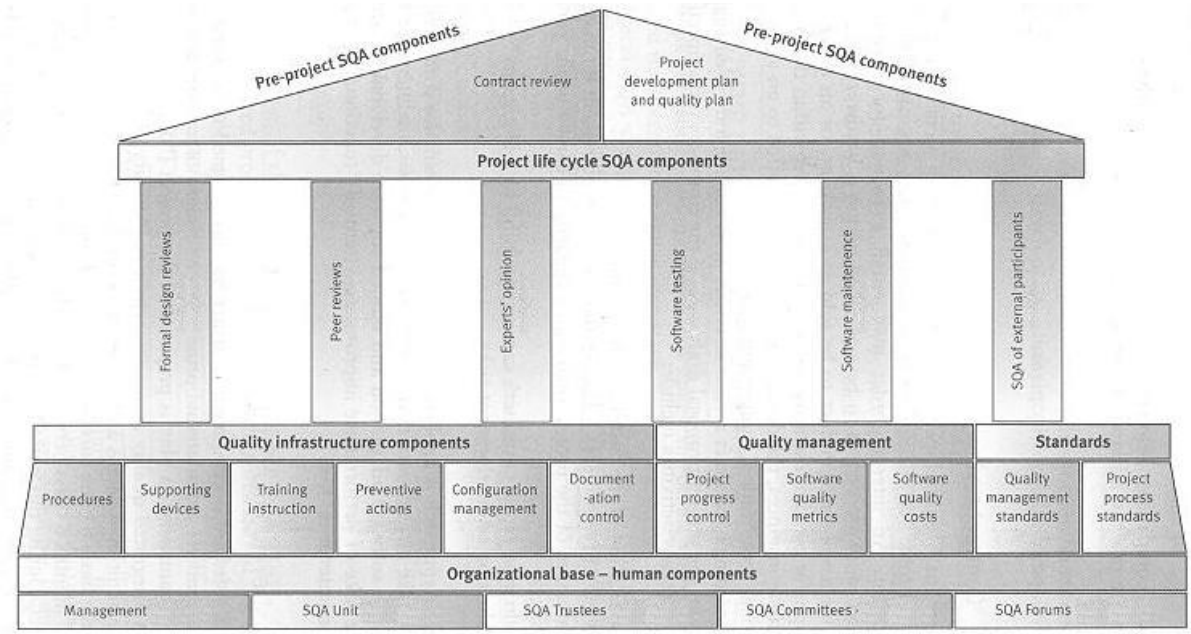


Figure 1: The SQA System Overview

1. Pre-project Components

The Pre-Project component ensures that the project has been adequately defined ensuring that the resources available, budget and so forth have not been misinterpreted by the client or the organisation. This improves the preliminary steps taken prior to initiating work on the project itself, thus leading to fewer errors later in the development phase.

There are two key concepts to this component;

1.1 Contract Review

The Contract Review begins the negotiations of a contract between the company and the client. One of the key focuses is on the areas or points that could go wrong, known as development risks.

The contract ensures that commitments have been documented including an agreement upon the functional specification, budget and the schedule.

The contract review activities must include a detailed examination of the project proposal draft

and the contract draft, each of which have different objectives, to ensure that the commitments have been properly and clearly defined. A checklist is often used by reviewers to make this stage easier and it is expected that the review stage will occur more than once.

The outcome from this stage is a documented agreement between the company and client stating the projects requirements, budgets and so on ensuring that no new changes have been added to the contract draft.

1.2 Development and Quality Plans

The development and Quality Plans stage occurs once an agreement has been made and a software development contract has been signed. The time between a proposal, the contract review and the signing of the contract could be prolonged and during this time changes may occur. Consequently proposal materials need to be revised and new plans are required; a development plan and a quality plan.

The development plan focuses on eleven specific elements, some of which are; the products of the project i.e. software products and design documents specifying deliverables. Project interfaces detailing what, if any, interfaces the product will have with existing software packages. The methodology clearly states which methodology should be used at each phase of the project.

The quality plan has four main elements; however, how many of these are used is dependent upon the project. Quality Goals refers to the goals of the product, it is always better to have more goals as it is easier to see how well the system performs. Planned review activities are a listing of all the planned reviews such as Design Reviews (DR) and Code Inspections. Planned software tests details all the software tests that are to be performed with details such as the unit, integration or complete system to be tested. Finally, planned acceptance tests for externally developed software; this is only necessary for products not developed in-house.

Approval of the two plans is necessary and is approved or rejected according to the procedures within the organisation.

2. *Project Lifecycle Activities and Assessment*

This component has two main stages; 1. The Development Lifecycle stage which aims to detect design and programming errors. 2. The Operation-Maintenance stage focuses on maintenance tasks that improve functionality.

When developing the product there are a number of activities that take place, these are reviews, such as formal design reviews and peer reviews, expert opinions, software testing, software maintenance and finally, the assurance of the quality of external participants' work which ensures that any efforts by

external members meet the quality and standards of the organisation.

The reviews occur during the design phase of the development process and should be conducted at any milestone. The formal Design Review (DR) is a very important document as the project cannot continue until a formal approval by the DR committee has been received. The document itself includes a list of action items (corrections) that are required. The peer review, guided by checklists, standards and past problems, is a document that reviews short documents or sections to attempt to detect design and programming faults documenting a list of its findings.

'Expert Opinions' is the use of an external member to review in-house work; it can reinforce the internal (in-house) quality assurance activities. Although not all organisations may use this approach it is useful for small organisations which may not have sufficient professional capabilities in-house or for a replacement to the regular in-house professional for whatever reasons. The outcome from this is often a document similar to a formal design review.

"Software Testing is a formal process carried out by a specialized team in which a software unit, several integrated software units or an entire software package are examined by running the programs on a computer. All associated tests are performed according to the approved test procedures on approved test cases"

D.Galin (2003)

In general, Software Testing is aimed at reviewing the running and functionality of the software in which the testing is based on a prepared list of test cases.

Software Maintenance specifies three categories in which maintenance for a system can fall into. The first is corrective maintenance in which the correction of failed software, for whatever reason, occurs. The second is adaptive maintenance in which maintenance occurs to the system it needs to be adapted; however the basic functionality of the system is left untouched. Finally, the third is functionality improvement where the system is modified to add improvements, this could significantly change the system compared to the other two points.

3. Infrastructure components for error prevention and Improvement

The main goal of this component is to reduce the number of errors in the system and improve productivity. This is achieved through the use of a number of sub-components.

1. Procedures and Work Instructions

A procedure describes how the process, a specific development activity, is performed whilst a work instruction is more specific and provides detailed directions for the use of methods. Having a

detailed guideline ensures that all members know how to achieve some goal.

"Work instructions and procedures contribute to the correct and effective performance of established technologies and routines"

http://kur2003.if.itb.ac.id/file/SW%20Quality%20Components.pdf

2. Supporting Quality Devices

'Quality Devices' are used to maximise efficiency and quality. A quality device could be a template or a checklist which saves time as the document will be complete and will not have to be developed, from scratch each time.

Using Quality Devices offers an improved form of communication and provides standardisation within the organisation as each document will be of the same nature.

3. Staff Training, Instruction and Certification

Staff training is a vital element to avoiding errors throughout the development of the system. Having well trained professional staff enables efficient and high quality performance from each member.

New staff have to be trained in respect to the standards and procedures of the organisation and existing staff should be re-trained when assignments change.

4. Preventive and Corrective Actions

Studying existing data for similar faults and failures can enable future ones to be solved easily either by correcting it once it has occurred or by preventing it from happening.

The data should be recorded, or found, in design review reports, software test reports or customer complaints. It should be managed effectively so that if it occurs in the future the solution, if one was found, can be easily accessible.

5. Configuration Management

This deals with the dangers of version releases. With intense work focusing on new versions and new software releases dangers arise when different members carry out the same tasks. This can lead to misidentification or the versions or releases or loss of records or development activities.

Configuration management introduces procedures that are used to control the change process and monitor it.

6. Documentation Control

Document Control is necessary to ensure long term availability of controlled documents. Controlled documents are maintained and updated. They are formally approved and contain evidence of the systems performance.

4. Software Quality Management

Quality management not only focuses on product quality but also the means in which it can be

achieved.

Some of the components used to support the managerial control of software development projects are the Project Progress Control, Software Quality Metrics and Software Quality costs.

The aim of Project Progress Control is simply to monitor the progress of the project to ensure that it does not deviate from its initial plans. It focuses particularly on monitoring resource usages, schedules (whether they are being met), risk management activities and the budget.

A Software Quality Metric is a measurement that is used to evaluate software quality in a system. The software is the input, and the output is a numeric value which represents the **degree to which the software possesses a given quality attribute.** The measures can apply to functional quality, productivity and the organization side of the project.

The Software Quality Costs are the costs that incur throughout the entire project; the total cost is calculated from the costs of control and the costs of failure. The organisations main interest is the sum of the total costs which will determine a success or failure.

5. SQA Standards, System Certification, and Assessment Components

This component focuses on using external tools to achieve the, in-house, goals of software quality assurance. There are three main objectives, D.Galin (2003):

1. Utilization of international professional knowledge
2. Improvement of coordination with other organizations quality systems
3. Objective professional evaluation and measurement of the achievements of the organizations quality systems

The standards available to achieve the above objectives can be classified into two sub-classes

1. Quality Management Standards – 'what' is required
2. Project Process Standards – 'how' it is done

The outcome of this component is simply to use external tools, i.e. staff, to achieve the desired in-house software quality assurance complying with the standards of the organization.

6. Organizing for SQA – The Human Components

SQA cannot be directly applied to an organization; instead an organizational base is required. The organizational base is collectively made up of the SQA Unit, SQA trustees, committees and forums along with the continuous support of the management.

The SQA Unit focuses purely on Software Quality Assurance, thus ensuring that all standards, procedures

and components are efficiently and correctly in use. This is, in part, done through the audits and quality programs that the SQA Unit is required to produce.

Management must ensure that all staff are aware of the quality policy, they are required to define sufficient resources and accurately assign an adequate number of staff to the tasks.

The SQA organizational base has three main objects

1. To aid the development and implementation of the SQA system
2. To detect and prevent deviations from organizational standards and procedures
3. To continuously suggest improvements that will benefit the SQA components

How to Design and Implement a Basic Quality Assurance Plan

A quality assurance plan should generally include two basic areas: how to address errors (quality-related events), and how to improve practice before an error occurs (continuous quality improvement). This document outlines steps to take in establishing a QA plan plan.

I. Design a means to effectively document quality-related events (QREs) and educate staff appropriately

1. Collect all relevant details of the event, identify the root cause(s), and make a plan to avoid the same error in the future (consider the example provided on the Board of Pharmacy's website)

2. Always educate staff on documented QREs and their resulting plans. 3. Many errors reported to the Board are due to poor customer service in resolving the issueconsider including training on how to handle an error as part of your plan

II. Identify one or two quality related parameters you would like to measure and improve. You might consider two categories of parameters:

1. Areas known to require improvement. a. These areas may be identified through a previous dispensing or procedural error, a deficiency notice from the Board, or observations of pharmacy staff. b. Monitoring will be with the intent to track successful

improvement.

2. Areas expected to be satisfactory a. These areas may be identified as perceived strengths in your pharmacy. b. The intent of monitoring may be to verify that processes are done correctly and to identify unsuspected weaknesses.

III. Design a method to measure the identified areas. Here are some tips:

1. Focus on quantitative measures that can show clear results

2. Utilize your computer system's capabilities where appropriate

3. Use random samples where appropriate (e.g. you don't necessarily have to go through the entire prescription log book to quantify counseling documentation)

4. Consider a method that can be accomplished in a reasonable amount of time by appropriate staff. Keep it simple.

5. Consider a method that can be done consistently as part of normal procedures.

6. Determine how often the measurement will be repeated and make plans to ensure it is not forgotten.

IV. Set appropriate goals

1. Perfection is not always a realistic goal. Determine what is acceptable for your practice.

2. Set an attainable goal and be prepared to update the goal when it is achieved.

3. Include instructions on what the person taking the measurement should do if the goal is not met (e.g. who to contact)

V. Be prepared to make new plans when goals are not met

1. Set a deadline for when unmet goals will be addressed 2. Be prepared to change policies or procedures in order to improve areas of deficiency

VI. Educate your staff on the Quality Assurance Plan, both at inception and at regular intervals. Include:

1. Why it is being done

2. What is being tracked

3. How to perform measurements

4. Progress in areas being monitored, including improvements implemented as a result thereof

5. Updates on any QREs, including the plan to avoid those errors in the future

VII. Quality assurance never ends

1. Continue to update your plan as necessary. Over time, the entire prescription process can be monitored and improved.

CMM and ISO

The ISO 9000 standards developed by the International Standards Organization are both concerned with quality and process management. The specific ISO standard of concern to software organizations is ISO 9001.

Questions frequently asked are:

- At what CMM level is an ISO compliant organiza- tion?
- Can a Level 2 or (3) organization be considered ISO compliant?
- Should SPI be based on CMM or ISO?

The ISO series of standards is a set of documents dealing with quality systems that can be used for external quality assurance purposes. They specify quality system requirements for use where a contract between two parties requires the demonstration of the supplier's capability to design and supply a product.The two parties could be an external client and a sup- plier, or they could both be internal.

ISO 9000 is a guideline that clarifies the distinctions and interrelationships between quality concepts and provides guidelines for the selection and use of a series of international standards on quality systems that can be used for internal quality management pur- poses (ISO 9004) and for external quality purposes (ISO 9001, 9002, and 9003).

The quality concepts addressed by these standards are:

- An organization should achieve and sustain the qual- ity of the product or service produced to continually meet the purchaser's stated or implied needs
- An organization should provide confidence to its own management that the intended quality is achieved
- An organization should provide confidence to the purchaser that the intended quality is being achieved in the delivered product or service provided

ISO 9001, "Quality systems-Model for quality assur- ance in desig.development, production, installation, and servicing," is the ISO standard that applies to software development and maintenance. There is a guideline, ISO 9000-3, for applying ISO 9001 to soft- ware processes. A British guide [TickIT] for 9001 provides additional guidelines on using ISO9000-3 and 9001 in the software area.

Mapping ISO 9001 to the CMM

Here, twenty clauses of ISO 9001 are mapped to practices of CMM

Clause 4.1 - Management responsibility

Addressed primarily by SQA and partly by SPP and SPTO (Level 2)

Clause 4.2 - Quality system

Addressed primarily by SQA and SPP (Level 2)

Clause 4.3 - Contract review

Addressed primarily by RM and SPP (Level 2)

Clause 4.4 - Design control

Addressed primarily by SPE, SPP, SPTO, and PR (Levels 2 and 3)

Clause 4.5 - Documentation and data control Addressed primarily by

SCM (Level)

Clause 4.6 - Purchasing

Addressed by SSM (Level 2)

Clause 4.7 - Control of customer-supplied product Addressed weakly by ISM (Level 3).

CMM change request written

Clause 4.8 - Product identification and traceability Addressed primarily by SCM and by SPE (Levels 2 and 3)

Clause 4.9 - Process control

Addressed by SPP, SPE, and SQA (Levels 2 and 3)

Clause 4.10 - Inspection and testing Addresses by SPE

and in PR (Level 3)

Clause 4.11 - Control of Inspection, Measuring, and Test Equipment

Addressed by SPE (Level 3)

Clause 4.12 - Inspection and test status Addressed by SPE and SCM (Levels 2 and 3)

Clause 4.13 - Control of nonconforming product Addressed by SPE and SCM (Levels 2 and 3)

Clause 4.14 - Corrective and preventive actions Addressed by SCM and SQA (Level 2)

Clause 4.15 - Handling storage, packaging, and pres- ervation delivery
Addressed partly by SCM, but actual delivery and installation not covered in present CMM (CMM change request written) (Level 2)

Clause 4.16 - Control of quality records
Addressed by SPE, SCM, and PR (Levels 2 and 3)

Clause 4.17 - Internal quality audits
Addressed by SQA (Level 2)

Clause 4.18 - Training
Addressed by TP (Level 3)

Clause 4.19 - Servicing
Not really addressed by CMM since maintenance is not a separate issue in CMM. Will be addressed in next version of CMM

Clause 4.20 - Statistical techniques
Practices described throughout CMM. Perhaps spe- cifically addressed by OPD, QPM, and SQM (Lev- els 3 and 4)

Contrasting ISO 9001 and CMM

Some issues in ISO 9001 are not covered in CMM, and vice versa. The levels of detail differ. Chapter 4 in ISO 9001 is 5 pages long, sections 5, 6, and 7 in ISO 9000-3 comprise 11 pages; CMM is over 500 pages long.

The ISO 9001 clauses with no strong relationship to CMM KPAs are control of customer-supplied prod- ucts and handling, packaging, preservation and deliv- ery

The clause in ISO 9001 that is addresses in CMM in a completely distributed fashion is servicing. There is significant debate about the exact relationships to CMM for corrective and preventive action and statis- tical techniques.

The biggest difference is the emphasis in CMM on continuous process improvement. ISO only addresses minimum criteria for an acceptable quality system.

CMM focuses strictly on software, while ISO 9001 has includes hardware, software, processed materials, and services.

For both CMM and ISO 9001, the bottom line is "Say what you do; do what you say."

Every Level 2 KPA is strongly related to ISO 9001 Every KPA is at least weakly related to ISO 9001

A CMM Level-1 organization can be ISO 9001 certi- fied; that organization would have significant Level-2 process strengths and noticeable Level-3 strengths.Given a reasonable implementation of the software process, a ISO 9001 certified organization should be at least close to CMM Level-2.

Can a CMM Level-3 organization be considered ISO 9001 compliant? Even a Level-3 organization would need to ensure that delivery and installation are addressed, but even a Level-2 organization would have comparatively little difficulty in obtaining ISO 9001 certification.

How ISO 9001 Fits into the Software

World (F. Coallier, *IEEE Software***, Janu- ary 1994)**

ISO 9001 has a strong emphasis on traditional manu- facturing quality control. It assumes products are pur- chased in a formal, contractual environment with detailed specifications that are correct. Such an envi- ronment is not the case for consumer or mass-market products, however, and it is naive to assume such conditions for complex products like those that incorporate software.

CMM/ISO 9001 -1

Software products are inherently complex and chal- lenging to scope, develop, implement, verify, validate, and maintain. This requires a total-quality approach focused on customer satisfaction and continuous improvement. In ISO 9001, continuous improvement is almost totally absent. It merely addresses the control of a nonconforming product and recommends corrective and preventive action.For an organization that develops and manufactures embedded software products, an ISO 9001 certifica- tion tells very little about its software development capability. Certification means only that some basic practices are in place.

CMM is a more comprehensive model to measure software development capability. It covers more pro- cesses and has a five-level rating system that empha- sizes continuous improvement.With ISO 9001, once you are certified, your challenge is only to maintain certification.

CMM can be used as a self-assessment. ISO 9001 certification requires auditors, which places emphasis on opinions of outsiders whose abilities may be unknown or marginal. ISO certification is usually prompted because certification is needed to get contracts. CMM review is usually done to improve and involves a more detailed study than does an ISO review.

ISO 9001 can still be worthwhile if: The auditors are good

- If the organization is CMM Level 1 or 2 because ISO 9001 covers the basics and can help the Organization grow.

The Malcolm Baldrige Criteria for Performance Excellence

In the early and mid-1980s, many industry and government leaders saw the need for a renewed emphasis on quality for doing business in an ever expanding, and more demanding, competitive world market. The Malcolm Baldrige National Quality Award was envisioned as a standard of excellence that would help U.S. organizations achieve world-class quality. The Malcolm Baldrige Criteria for Performance Excellence have played a major role in achieving the goals established for the Baldrige Award. They now are accepted widely, not only in the United States but also around the world, as the standard for performance excellence.

For over 20 years, the Baldrige Criteria have been used by thousands of U.S. organizations to stay abreast of ever-increasing competition and to improve performance. In today's business, health care, education, nonprofit, and government environments, the Criteria help organizations respond to current challenges: openness and transparency in governance and ethics; the need to create value for the business and its customers, patients, or students; and the challenges of rapid innovation and capitalizing on knowledge assets. Whether an organization is small or large, is for-profit or not-for- profit, or has one location or multiple sites across the globe, the Criteria provide a valuable framework that can help plan and achieve in an uncertain environment. The Criteria help to assess performance on a wide range of key business indicators: customer, product and service, financial, human resource, and operational. The Criteria can help to align resources and approaches, such as ISO9000, Lean Enterprise, Balanced Scorecard, Six Sigma, and regulatory requirements; improve communication, productivity, and effectiveness; and achieve strategic goals.

The Criteria are built upon a set of interrelated core values and concepts found in high-performing organizations. These core values and concepts are embodied in seven linked categories. Together they provide the foundation for an organization to integrate key business requirements within a results-oriented framework to create a basis for action and feedback.

The Criteria for Performance Excellence Fact Sheet

What are the Baldrige criteria?

The Baldrige Criteria for Performance Excellence are a framework that any organization can use to improve overall performance. While the Criteria characteristics, goals, and purposes remain constant, the Criteria have evolved significantly over time to help organizations address current economic and marketplace challenges and opportunities.

The Criteria Characteristics:

- Focus on results in all areas of organizational performance to ensure that all strategies are balanced.
- Are non-prescriptive and adaptable to promote creative and flexible approaches for meeting requirements, and to foster incremental and breakthrough improvements.
- Support a systems perspective to maintain organization-wide goal alignment.
- Support goal-based diagnosis on a profile of performance oriented strengths and opportunities for improvement.

Criteria Goals: The Criteria are designed to help organizations use an integrated approach to organizational performance management that results in

- Delivery of ever-improving value to customers, contributing to marketplace success
- Improvement of overall organizational effectiveness and capabilities
- Organizational and personal learning

Criteria Purposes: The criteria are used by thousands of organizations of all kinds for self-assessment and training and as a tool to develop performance and business processes. For many organizations, using the criteria results in better employee relations, higher productivity, greater customer satisfaction, increased market share, and improved profitability. According to a report by the Conference Board, a business membership organization, –A majority of large U.S. firms have used the criteria of the Malcolm Baldrige National Quality Award for self-improvement, and the evidence suggests a long-term link between use of the Baldrige Criteria and improved business performance.‖

In addition, the Criteria have three important roles in strengthening U.S. competitiveness

- To help improve organizational performance practices, capabilities, and results
- To facilitate communication and sharing of best practices information among U.S. organizations of all types
- To serve as a working tool for understanding and managing performance and for guiding organizational planning and opportunities for learning

.

Seven categories make up the award criteria:

- **Leadership**—Examines how senior executives guide and sustain the organization and how the organization addresses Governance, ethical, legal and community responsibilities.
- **Strategic planning**—Examines how the organization sets strategic directions and how it determines and deploys key action plans.
- **Customer focus**—Examines how the organization determines requirements and

expectations of customers and markets; builds relationships with customers; and acquires, satisfies, and retains customers.

- **Measurement, analysis, and knowledge management**—Examines the management, use, analysis, and improvement of data and information to support key organization processes as well as how the organization reviews its performance.
- **Workforce focus**—Examines how the organization engages, manages, and develops all those actively involved in accomplishing the work of the organization to develop full potential and how the workforce is aligned with the organization's objectives.
- **Process management**—Examines aspects of how key production/delivery and support processes are designed, managed, and improved.
- **Results**—Examines the organization's performance and improvement in its key business areas: customer satisfaction, financial and marketplace performance, workforce, product/service, and operational effectiveness, and leadership. The category also examines how the organization performs relative to competitors.

Core Values and Concepts: The Criteria are built on a set of interrelated, embedded beliefs and behaviors found in high-performing organizations. The core values and concepts are the foundation for integrating key business requirements within a results-oriented framework that creates a basis for action and feedback.

Why Baldrige?

Why should an organization consider using the Baldrige Criteria?

Baldrige is the most comprehensive management framework available. It enables leaders to understand all of the internal and external forces that drive their business; to prioritize, enhance, and improve what is critical to success; and to select the courses of action that achieve, increase, and sustain the best possible overall performance. In short, implementing the Baldrige Criteria guide organizations to do the right things, at the right time, and in the right way.

Baldrige works for all types and sizes of organizations because it asks the questions that all high performing organizations consider and leaves the answers to those who can best determine them – the people who work in the organization. The knowledge gained promotes creativity and flexibility to order to deliver ever-improving value to customers and improve organizational effectiveness and capabilities.

Baldrige works if:

1) Leaders and the organization have the willingness and ability to develop an organizational culture based on the 11 core values of high performing organizations defined by the Criteria.
2) Leaders and the organization are willing to commit to a long-term journey of continuous learning and improvement.

What are the benefits compared to other tools and management systems?

Baldrige has a true systems perspective – it looks at all components of an organization with equal emphasis and focuses on how each part impacts and links with the others. It helps leaders align and integrate their leadership, strategy, customer & market focus, data analysis & knowledge management, workforce, and process management systems to produce the best overall results. Most other tools and management systems focus on one or a few of these components more than the rest. Other tools and management systems complement Baldrige and can provide more detailed guidance on ‑how‖ to implement than Baldrige does. Using Baldrige as your management system will help you determine which of these tools will most benefit your organization and when.

CMMI Representation

CMMI enables you to approach process improvement because it provides the latest best practices for product and service development and maintenance. The CMMI best practices enable organizations to do the following: • More explicitly link management and engineering activities to their business objectives • Expand the scope of and visibility into the product lifecycle and engineering activities to ensure that the producer or service meets customer expectations • Incorporate lessons learned from additional areas of best practice (e.g.,

measurement, risk management, and supplier management) • Implement more robust high maturity practices • Address additional organizational functions critical to their products and services • More fully comply with relevant ISO standards

Levels of the Capability Maturity model

There are five levels defined along the continuum of the CMM, and, according to the SEI: "Predictability, effectiveness, and control of an organization's processes are believed to improve as the organization moves up these five levels. While not rigorous, the empirical evidence to date supports this belief."

Level 1 Initial It is characteristic of processes at this level that they are (typically) undocumented and in a state of dynamic change, tending to be driven in an ad hoc, uncontrolled and reactive manner by users or events. This provides a chaotic or unstable environment for the processes.

Level 2 – Managed It is characteristic of processes at this level that some processes are repeatable, possibly with consistent results. Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.

Level 3 – Defined It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time. These standard processes are in place (i.e., they are the AS-IS processes) and used to establish consistency of process performance across the organization.

Level 4- Quantitatively Managed It is characteristic of processes at this level that, using process metrics, management can effectively control the AS IS process (e.g., for software development). In particular, management can identify way to adjust and adapt the process to particular projects without measurable losses of quality or deviations from specifications. Process capability is established from this level.

Level 5- Optimizing It is a characteristic of processes at this level that the focus is on continually

improving process performance through both incremental and innovative technological changes/ improvements. At maturity level 5, processes are concerned with addressing statistical common causes of process (for example, shifting the mean of the process performance) to improve process performance. This would be done at the same time as

maintaining the likelihood of achieving the established quantitative process improvement objectives.

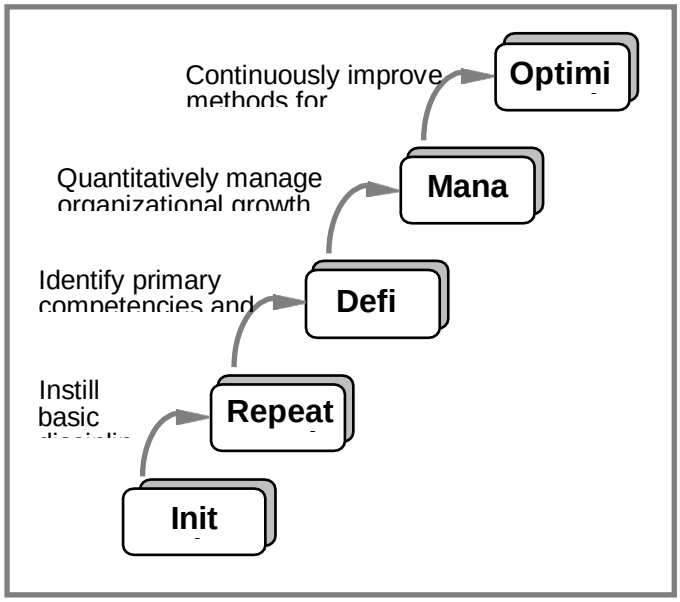
Definition of the P-CMM Maturity Levels

As a capability maturity model, the P-CMM guides organizations in establishing and improving their workforce practices through five evolutionary stages. Achieving each maturity level in the P-CMM institutionalizes new capabilities for developing the knowledge and skills of the workforce, resulting in an overall increase in the talent of the organization. Growth through the maturity levels creates fundamental changes in how people are developed and organized and in their working culture.

Figure 2.1 depicts the five maturity levels of the P-CMM. Each maturity level provides a layer in the foundation for continuous improvement of an organization's workforce practices. In maturing from the Initial to the Repeatable level, the organization installs the discipline of performing the basic practices. In maturing to the Defined level, these practices are tailored to enhance the particular knowledge, skills, and work methods that best support the organization's business. In maturing to the Managed level, the organization develops competency-based, high-performance teams and empirically evaluates how effectively its workforce practices are meeting objectives. In maturing to the Optimizing level, the organization looks continually for innovative ways to improve its workforce capability and to support individuals in their pursuit of professional excellence.

Level 1 - The Initial Level

At the Initial level, the performance of workforce activities is inconsistent. The organization typically provides forms for activities such as performance appraisals or position requisitions, but offers little guidance or training in conducting the activities supported by these forms. Typically managers have not been trained in performing most of their workforce responsibilities, so their ability to manage those who report to them is based on previous experience and their personal "people skills." These organizations are not necessarily abusive or inconsiderate. Their problem is that they do not have the ability to



systematically develop the competitive capability of their workforce.

Figure 2.1 The Five Maturity Levels of the P-CMM

In the worst circumstances, managers in Level 1 organizations do not accept developing the members of their unit as a primary personal responsibility. They perform workforce activities such as interviewing job candidates or conducting performance appraisals with little preparation, often resulting in poor staffing decisions or disgruntled employees. The human resources department too often imports practices and applies them with little analysis of their effectiveness. Individuals in most Level 1 organizations do not take workforce practices seriously, since they do not believe the practices have much relation to their real work and level of contribution to the organization.

The workforce capability of a Level 1 organization is unknown, since there is little effort to measure or improve it. Individuals are motivated to pursue their own agendas, since there are few incentives in place to align their motivations with the business objectives of the organization. Turnover is high when people feel there are better working conditions or growth potential in another organization. Consequently, the level of knowledge and skills available in the organization does not grow over time because of the need to replace experienced and knowledgeable individuals who have left the organization.

Level 2 - The Repeatable Level

The primary objectives at the Repeatable level are to eliminate problems that keep people from being able to perform their work responsibilities effectively and to establish a foundation of workforce practices that can be continuously improved in developing the workforce. The most frequent problems that keep people from being able to perform effectively in low- maturity organizations include

- environmental distractions
- unclear performance objectives
- lack of relevant knowledge or skill
- poor communication

In maturing to the Repeatable level, an organization establishes policies that commit it to developing its people. A primary objective in achieving a repeatable capability is to establish a sense of responsibility and discipline in performing basic workforce practices. These practices ensure that the people in each unit will have the knowledge and skills required to perform their current assignment. When these practices are institutionalized, the organization has laid a foundation on which it can build improved methods and practices.

At the Repeatable level, those who have been assigned responsibility for

performing workforce activities accept personal responsibility for ensuring that all workforce practices are implemented effectively. In doing so, they accept the growth and development of their staff as a primary responsibility of their position. When people take their workforce responsibilities seriously, they begin to develop repeatable methods for performing specific activities such as interviewing or establishing performance criteria. Individuals will notice greater consistency in the performance of workforce functions within their group, although different managers or groups may have individual variations in the specific methods they use.

The effort to implement improved workforce practices begins when executive management commits the organization to constantly improve the knowledge, skills, motivation, and performance of its workforce. The organization states that the continuous development of its workforce is a core value. The organization documents policies and develops basic workforce practices that the units will implement. Units develop plans for satisfying their workforce needs and responsibilities. These initial needs are in the areas of the work environment, communication, staffing, performance management, training, and compensation. Until these basic workforce practices become institutionalized, the organization will have difficulty adopting more sophisticated workforce practices.

Level 3 - The Defined Level

Organizations at the Repeatable level find that although they are performing basic workforce practices, there is inconsistency in how these practices are performed across units. The organization is not capitalizing on opportunities to standardize its best workforce practices, because it has not identified the common knowledge and skills needed across its units and the best practices to be used for developing them. The organization is motivated to achieve the Defined level in order to gain a strategic competitive advantage from its core competencies.

At the Defined level, the organization begins to adapt its workforce practices to the specific nature of its business. By analyzing the skills required by its workforce and the

business functions they perform, the organization identifies the core competencies required to perform its business. The organization then adapts its workforce practices to develop the specific knowledge and skills that compose these core competencies. The organization identifies best practices in its own workforce activities or those of other organizations and tailors them as the basis for adapting its workforce practices.

The organization analyzes its business processes to determine the core competencies involved in its work and the knowledge and skills that constitute these competencies. The organization then develops strategic and near-term plans for developing these competencies across the organization. A program is defined for systematically developing core competencies, and individuals' career development strategies are planned to support competency development for each individual. The organization administers its workforce practices to develop and reward growth in its core competencies and to apply them to improve performance.

A common organizational culture can develop at the Defined level, because the organization becomes focused on developing and rewarding a set of core competencies. This culture places importance on growing the organization's capability in its core competencies, and the entire workforce begins sharing responsibility for this growth. Such a culture is reinforced when workforce practices are adapted to encourage and reward growth in the organization's core competencies. This culture can be enhanced by establishing a participatory environment where individuals and groups are involved in decisions regarding their work.

The workforce capability of organizations at the Defined level is based on having a workforce that possesses the basic knowledge and skills to perform the core business functions of the organization. Knowledge and skills in the organization's core competencies are more evenly spread across the organization. The organization has improved its ability to predict the performance of its work activities based on knowing the level of knowledge and skills available in its workforce. Also, it has established a foundation on which continuous development of knowledge and skills can be built.

Level 4 - The Managed Level

Organizations at the Defined level have established the foundation for continuously improving their workforce. At the Managed level, the organization takes the first steps in capitalizing on managing its core competencies as a strategic advantage. It sets quantitative objectives for growth in core competencies and for the alignment of performance across the individual, team, unit, and organizational levels. These measures establish the quantitative foundation for evaluating trends in the capability of the organization's workforce. Further, it seeks to maximize the effectiveness of applying these competencies by developing teams that integrate complementary knowledge and skills.

At the Managed level, high-performance teams composed of people with complementary knowledge and skills are developed where conditions support their functioning. Team-building activities are performed to improve the effectiveness of these teams. When applied to teams,Workforce practices are tailored to support team development and performance.

Mentors are made available to both individuals and teams. Mentors use their experience to provide personal support, guidance, and some skill development. Mentors also provide another way to retain and disseminate lessons learned across the organization. Organizational growth in each of the organization's core competencies is quantitatively managed. Data on the level of core competencies in the organization are analyzed to determine trends and capability. These competency trends are then used to evaluate the effectiveness of competency-related workforce practices. In addition, performance data are collected and analyzed for trends in the alignment of performance at the individual, team, unit, and organizational levels. Trends in the alignment of performance are used to evaluate the effectiveness of performance-related workforce practices. These trends are tracked against the objectives set in the strategic and near-term workforce plans.

The workforce capability of Level 4 organizations is predictable because the current capability of the workforce is known quantitatively. The organization has also developed a mechanism for deploying its competencies effectively through high-performance, competency-based teams. Future trends in workforce capability and performance can be predicted because the capability of the workforce practices to improve the knowledge and skills of the workforce is known quantitatively. This level of workforce capability provides

the organization with an important predictor of trends in its business capability.

Level 5 - The Optimizing Level

At the Optimizing level, there is a continuous focus on improving individual competencies and finding innovative ways to improve workforce motivation and capability. The organization supports individuals' effort toward continuous development of personal competencies. Coaches are provided to support further development of personal or team competencies.

Data on the effectiveness of workforce practices are used to identify needs for innovative workforce practices or technologies. Innovative practices and technologies are evaluated and the most promising are used in exploratory trials. Successful innovations are then transferred into use throughout the organization.

The workforce capability of Optimizing organizations is continuously improving because they are perpetually improving their workforce practices. Improvement occurs both by incremental advancements in their existing workforce practices and by adoption of innovative practices and methods that may have a dramatic impact. The culture created in an Optimizing organization is one in which all members of the workforce are striving to improve their own, their team's, and their unit's knowledge, skills, and motivation in order to improve the organization's overall performance. The workforce practices are honed to create a culture of performance excellence.

Six Sigma

Six Sigma is one of the most popular quality methods lately. It is the rating that signifies "best in class," with only 3.4 defects per million units or operations (DPMO). Its concept works and results in remarkable and tangible quality improvements when implemented wisely. Today, Six Sigma processes are being executed in a vast array of organization and in a wide variety of functions. Fueled by its success at large companies such as Motorola, General electric, Sony, and Allied Signal, the methodology is proving to be much than just a quality initiative.

Why are these large companies embracing Six Sigma? What makes this methodology different from the others?

The goal of Six Sigma is not to achieve six sigma levels of quality, but to improve profitability. Prior to Six Sigma, improvements brought about by quality programs, such as Total Quality Management (TOM) and ISO 9000, usually had no visible impact on a company's net income. In general, the consequences of immeasurable improvement and invisible impact caused these quality programs gradually to be. Six Sigma was originally developed as a set of practices designed to improve manufacturing processes and eliminate defects, but its application was subsequently extended to other types of business processes as well In Six Sigma, a defect is defined as anything that lead to customer dissatisfaction.

• Six Sigma stands for six standard deviation from mean (sigma is thee Greek letter used to represent standard deviation in statistics).

• Six Sigma methodologies provide the techniques and tools to improve the capability and reduce the defects in any process.

• Six Sigma strives for perfection. It allows for only 3.4 defects per million opportunities (or 99.999666 percent accuracy)

• Six Sigma improves the process performance decrease variation and maintains consistent quality of the process output. This leads to defect reduction and improvements in profits, product quality and customer satisfaction.

• Six Sigma incorporates the basic principles and techniques used in business, statistics and engineering.

• The objective of Six Sigma principle is to achieve zero defects products/ process. It allows and engineering.

• The objective of Six Sigma principle is to achieve zero defects products/ process. It allows 4.4 defects per million opportunities.

Features that St Six Sigma

Apart from previous quality improvement initiatives include— • A clear focus on achieving measurable and quantifiable financial returns from any Six Sigma project. • An increased emphasis on strong and passionate management leadership and support. • A special infrastructure of " Champions, " " Master black Belts, " "black Belts, " etc. to lead and implement the Six Sigma approach. • A clear commitment to making decisions on the basis of verifiable data, rather than assumptions and guesswork.

Sigma Levels

• 1 sigma = 690, 000 DPMO= 31% efficiency • 2 sigma = 308,000 DPMO= 69.2% efficiency • 3 sigma = 66,800 DPMO= 93.32% efficiency • 4 sigma= 6,210 DPMO = 99.379 % efficiency • 5 sigma = 230 DPMO = 99.977 % efficiency • 6 sigma = 3.4 DPMO = 99.9997 % efficiency

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

School of Computing

Department of Computer Science and Engineering

and

Department of Information Technology

SCS1608 - Software Quality Assurance and Testing

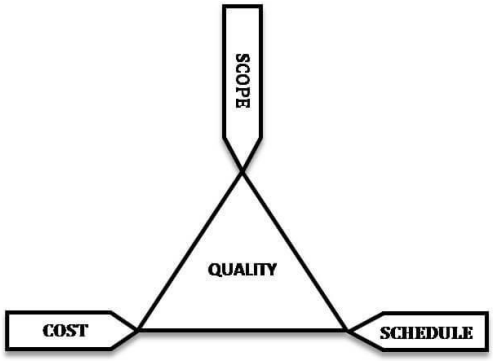
UNIT - II

UNIT II

1.What is Software Testing Metric?

Software Testing Metric is be defined as a quantitative measure that helps to estimate the progress, quality, and health of a software testing effort. A Metric defines in quantitative terms the degree to which a system, system component, or process possesses a given attribute.

The ideal example to understand metrics would be a weekly mileage of a car compared to its ideal mileage recommended by the manufacturer.



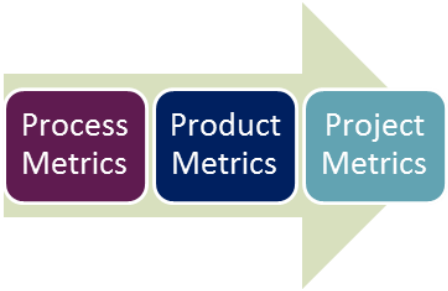
Software testing metrics - Improves the efficiency and effectiveness of a software testing process.

Software testing metrics or software test measurement is the quantitative indication of extent, capacity, dimension, amount or size of some attribute of a process or product.

2.Why Test Metrics are Important?

- "We cannot improve what we cannot measure" and Test Metrics helps us to do exactly the same.
- Take decision for next phase of activities
- Evidence of the claim or prediction
- Understand the type of improvement required
- Take decision or process or technology change

3.Software metrics can be classified into three categories –



Product metrics − Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.

Process metrics − These characteristics can be used to improve the development and maintenance activities of the software.

Project metrics − This metrics describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.

4.Software quality metrics

are a subset of software metrics that focus on the quality aspects of the product, process, and project. These are more closely associated with process and product metrics than with project metrics.

Software quality metrics can be further divided into three categories −

I. Product quality metrics

II. In-process quality metrics

III. Maintenance quality metrics

I. Product Quality Metrics

This metrics include the following −

- Mean Time to Failure

- Defect Density
- Customer Problems
- Customer Satisfaction

Mean Time to Failure

It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics, and weapons.

Defect Density

It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit. This metric is used in many commercial software systems.

Customer Problems

It measures the problems that customers encounter when using the product. It contains the customer's perspective towards the problem space of the software, which includes the non-defect oriented problems together with the defect problems.

The problems metric is usually expressed in terms of **Problems per User-Month (PUM)**.

PUM = Total Problems that customers reported (true defect and non-defect oriented problems) for a time period + Total number of license months of the software during the period

Where,

Number of license-month of the software = Number of install license of the software × Number of months in the calculation period

PUM is usually calculated for each month after the software is released to the market, and also for monthly averages by year.

Customer Satisfaction

Customer satisfaction is often measured by customer survey data through the five-point scale −

Very satisfied

Satisfied

Neutral

Dissatisfied

Very dissatisfied

Satisfaction with the overall quality of the product and its specific dimensions is usually obtained through various methods of customer surveys. Based on the five-point-scale data, several metrics with slight variations can be constructed and used, depending on the purpose of analysis. For example −

Percent of completely satisfied customers

Percent of satisfied customers

Percent of dis-satisfied customers

Percent of non-satisfied customers

Usually, this percent satisfaction is used.

II. In-process Quality Metrics

In-process quality metrics deals with the tracking of defect arrival during formal machine testing for some organizations. This metric includes −

- Defect density during machine testing
- Defect arrival pattern during machine testing
- Phase-based defect removal pattern
- Defect removal effectiveness

Defect density during machine testing

Defect rate during formal machine testing (testing after code is integrated into the system library) is correlated with the defect rate in the field. Higher defect rates found during testing is an indicator that the software has experienced higher error injection during its development process, unless the higher testing defect rate is due to an extraordinary testing effort.

This simple metric of defects per KLOC or function point is a good indicator of quality, while the software is still being tested. It is especially useful to monitor subsequent releases of a product in the same development organization.

Defect arrival pattern during machine testing

The overall defect density during testing will provide only the summary of the defects. The pattern of defect arrivals gives more information about different quality levels in the field. It includes the following −

The defect arrivals or defects reported during the testing phase by time interval (e.g., week). Here all of which will not be valid defects.

The pattern of valid defect arrivals when problem determination is done on the reported problems. This is the true defect pattern.

The pattern of defect backlog overtime. This metric is needed because development organizations cannot investigate and fix all the reported problems immediately. This is a workload statement as well as a quality statement. If the defect backlog is large at the end of the development cycle and a lot of fixes have yet to be integrated into the system, the stability of the system (hence its quality) will be affected. Retesting (regression test) is needed to ensure that targeted product quality levels are reached.

Phase-based defect removal pattern

This is an extension of the defect density metric during testing. In addition to testing, it tracks the defects at all phases of the development cycle, including the design reviews, code inspections, and formal verifications before testing.

Because a large percentage of programming defects is related to design problems, conducting formal reviews, or functional verifications to enhance the defect removal capability of the process at the front-end reduces error in the software. The pattern of phase-based defect removal reflects the overall defect removal ability of the development process.

With regard to the metrics for the design and coding phases, in addition to defect rates, many development organizations use metrics such as inspection coverage and inspection effort for in-process quality management.

Defect removal effectiveness

It can be defined as follows −

DRE=DefectremoveddduringadevelopmentphaseDefectslatentintheproduct×100%

This metric can be calculated for the entire development process, for the front-end before code integration and for each phase. It is called **early defect removal** when used for the front-end and **phase effectiveness** for specific phases. The higher the value of the metric, the more effective the development process and the fewer the defects passed to the next phase or to the field. This metric is a key concept of the defect removal model for software development.

III. Maintenance Quality Metrics

Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

- Fix backlog and backlog management index
- Fix response time and fix responsiveness
- Percent delinquent fixes
- Fix quality

Fix backlog and backlog management index

Fix backlog is related to the rate of defect arrivals and the rate at which fixes for reported problems become available. It is a simple count of reported problems that remain at the end of each month or each week. Using it in the format of a trend chart, this metric can provide meaningful information for managing the maintenance process.

Backlog Management Index (BMI) is used to manage the backlog of open and unresolved problems.

BMI=(NumberofproblemsclosedduringthemonthNumberofproblemsarriveddduringthemonth)×100%

If BMI is larger than 100, it means the backlog is reduced. If BMI is less than 100, then the backlog increased.

Fix response time and fix responsiveness

The fix response time metric is usually calculated as the mean time of all problems from open to close. Short fix response time leads to customer satisfaction.

The important elements of fix responsiveness are customer expectations, the agreed-to fix time, and the ability to meet one's commitment to the customer.

Fix Quality

Fix quality or the number of defective fixes is another important quality metric for the maintenance phase. A fix is defective if it did not fix the reported problem, or if it fixed the original problem but injected a new defect. For mission-critical software, defective fixes are detrimental to customer satisfaction. The metric of percent defective fixes is the percentage of all fixes in a time interval that is defective.

A defective fix can be recorded in two ways: Record it in the month it was discovered or record it in the month the fix was delivered. The first is a customer measure; the second is a process measure. The difference between the two dates is the latent period of the defective fix.

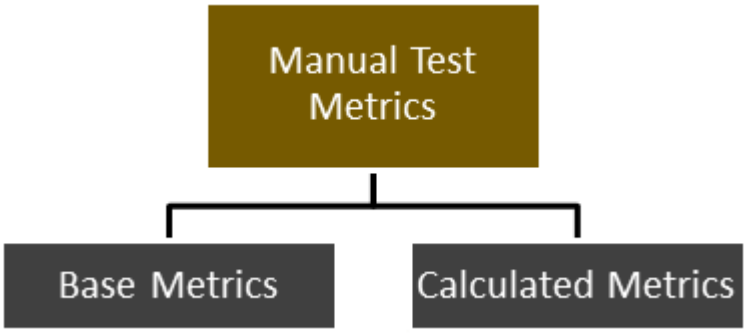
Usually the longer the latency, the more will be the customers that get affected. If the number of defects is large, then the small value of the percentage metric will show an optimistic picture. The quality goal for the maintenance process, of course, is zero defective fixes without delinquency.

5.Manual Test Metrics

In Software Engineering, Manual test metrics are classified into two classes

Base Metrics

Calculated Metrics



Base metrics is the raw data collected by Test Analyst during the test case development and execution (**# of test cases executed, # of test cases**). While calculated metrics are derived from the data collected in base metrics. Calculated metrics is usually followed by the test manager for test reporting purpose (**% Complete, % Test Coverage**).

Depending on the project or business model some of the important metrics are

Test case execution productivity metrics

Test case preparation productivity metrics

Defect metrics

Defects by priority

Defects by severity

Defect slippage ratio

6.Test Metrics Life Cycle

Different stages of Metrics life cycle	**Steps during each stage**
Analysis	Identification of the Metrics Define the identified QA Metrics
Communicate	Explain the need for metric to stakeholder and testing team

	Educate the testing team about the data points to need to be captured for processing the metric
Evaluation	Capture and verify the data Calculating the metrics value using the data captured
Report	Develop the report with an effective conclusion Distribute the report to the stakeholder and respective representative Take feedback from stakeholder

How to calculate Test Metric

Sr#	Steps to test metrics	Example
1	Identify the key software testing processes to be measured	Testing progress tracking process
2	In this Step, the tester uses the data as a baseline to define the metrics	The number of test cases planned to be executed per day
3	Determination of the information to be followed, a frequency of tracking and the person responsible	The actual test execution per day will be captured by the test manager at the end of the day
4	Effective calculation, management, and interpretation of the defined metrics	The actual test cases executed per day

5	Identify the areas of improvement depending on the interpretation of defined metrics	The Test Case execution falls below the goal set, we need to investigate the reason and suggest the improvement measures

Example of Test Metric

To understand how to calculate the test metrics, we will see an example of a percentage test case executed.

To obtain the execution status of the test cases in percentage, we use the formula.

Percentage test cases executed= (No of test cases executed/ Total no of test cases written) X 100

Likewise, you can calculate for other parameters like **test cases not executed, test cases passed, test cases failed, test cases blocked, etc.**

Test Metrics Glossary

Rework Effort Ratio = (Actual rework efforts spent in that phase/ total actual efforts spent in that phase) X 100

Requirement Creep = (Total number of requirements added/No of initial requirements)X100

Schedule Variance = (Actual efforts – estimated efforts) / Estimated Efforts) X 100

Cost of finding a defect in testing = (Total effort spent on testing/ defects found in testing)

Schedule slippage = (Actual end date – Estimated end date) / (Planned End Date – Planned Start Date) X 100

P**assed Test Cases Percentage** = (Number of Passed Tests/Total number of tests executed) X 100

Failed Test Cases Percentage = (Number of Failed Tests/Total number of tests executed) X 100

Blocked Test Cases Percentage = (Number of Blocked Tests/Total number of tests executed) X 100

Fixed Defects Percentage = (Defects Fixed/Defects Reported) X 100

Accepted Defects Percentage = (Defects Accepted as Valid by Dev Team /Total Defects Reported) X 100

Defects Deferred Percentage = (Defects deferred for future releases /Total Defects Reported) X 100

Critical Defects Percentage = (Critical Defects / Total Defects Reported) X 100

Average time for a development team to repair defects = (Total time taken for bugfixes/Number of bugs)

Number of tests run per time period = Number of tests run/Total time

Test design efficiency = Number of tests designed /Total time

Test review efficiency = Number of tests reviewed /Total time

Bug find rote or Number of defects per test hour = Total number of defects/Total number of test hours

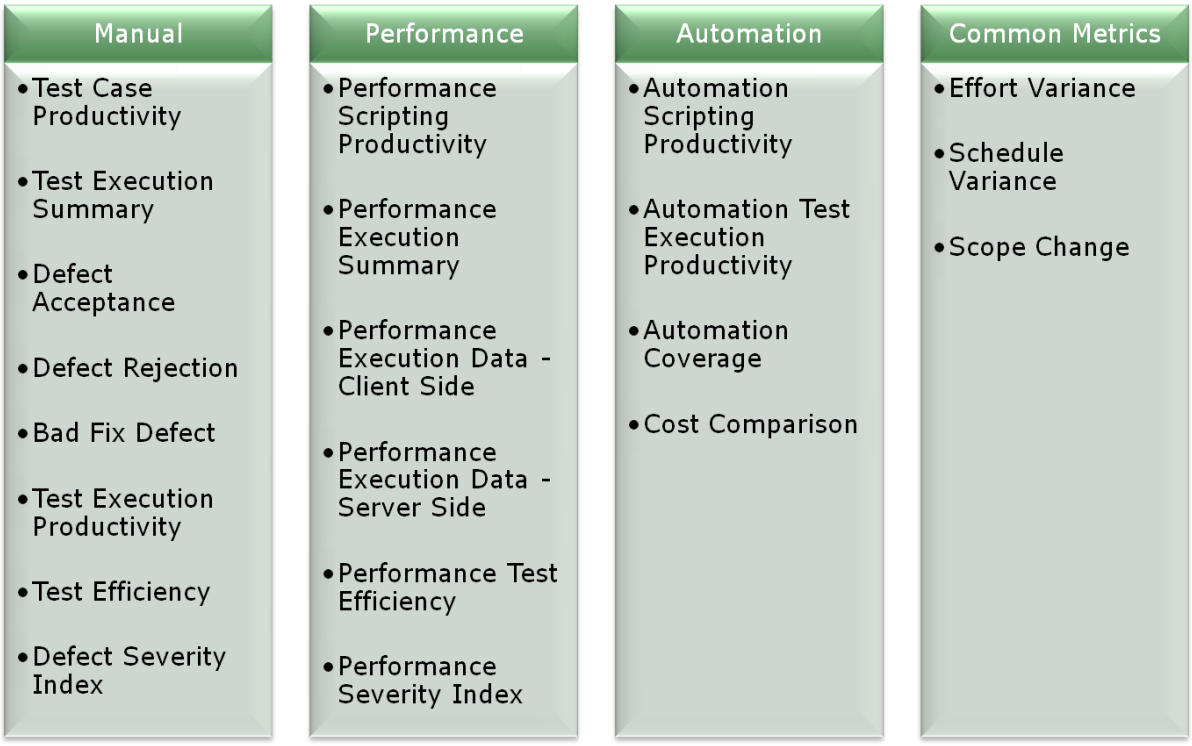
7.Type of Software Testing Metrics

Based on the types of testing performed, following are the types of software testing metrics:

-

1. Manual Testing Metrics
2. Performance Testing Metrics
3. Automation Testing Metrics

Following figure shows different software testing metrics.



Let's have a look at each of them.

7.1Manual Testing Metrics

Test Case Productivity (TCP)

This metric gives the test case writing productivity based on which one can have a conclusive remark.

$$Test\ Case\ Productivity = \left[\frac{Total\ Raw\ Test\ Steps}{Efforts\ (hours)}\right] Step(s)/hour$$

Example

Test Case Name	Raw Steps
XYZ_1	30
XYZ_2	32
XYZ_3	40
XYZ_4	36
XYZ_5	45
Total Raw Steps	**183**

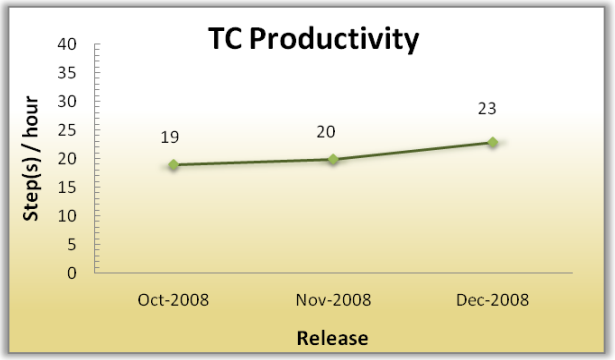
Efforts took for writing 183 steps is 8 hours.

TCP=183/8=22.8

Test case productivity = 23 steps/hour

One can compare the Test case productivity value with the previous release(s) and draw the most effective conclusion from it.

TC Productivity Trend

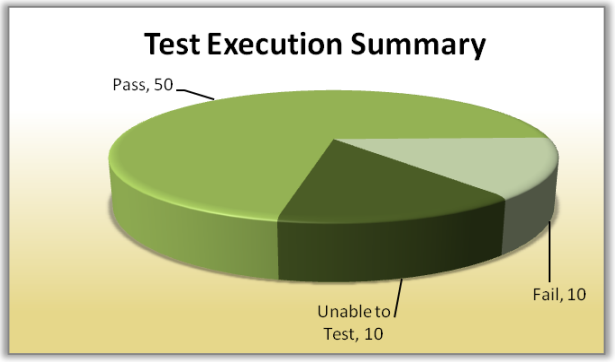


Test Execution Summary

This metric gives classification of the test cases with respect to status along with reason, if available, for various test cases. It gives the statical view of the release. One can collect the data for the number of test case executed with following status: -

- Pass.
- Fail and reason for failure.
- Unable to Test with reason. Some of the reasons for this status are time crunch, postponed defect, setup issue, out of scope.

Summary Trend



One can also show the same trend for the classification of reasons for various unable to test and fail test cases.

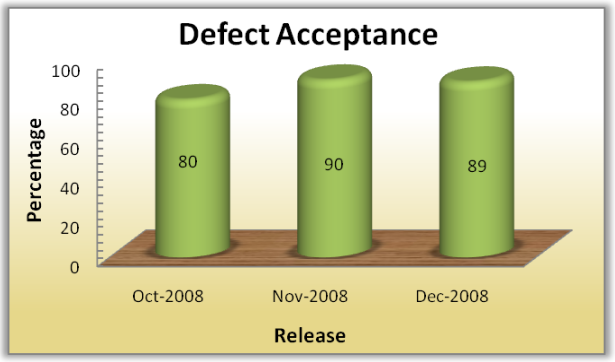
Defect Acceptance (DA)

This metric determine the number of valid defects that testing team has identified during execution.

$$Defect\ Acceptance = \left[\frac{Number\ of\ Valid\ Defects}{Total\ Number\ of\ Defects} * 100 \right] \%$$

The value of this metric can be compared with previous release for getting better picture

Defect Acceptance Trend



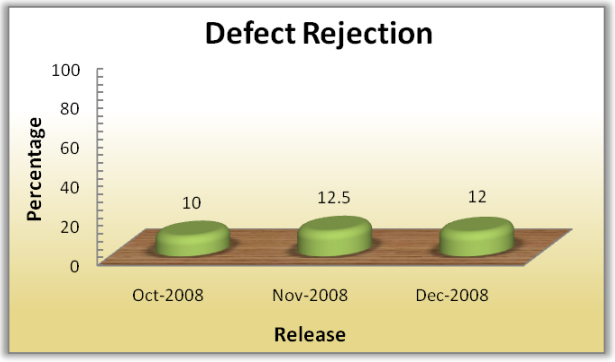
Defect Rejection (DR)

This metric determine the number of defects rejected during execution.

$$Defect\ Rejection = \left[\frac{Number\ of\ Defect(s)\ Rejected}{Total\ Number\ of\ Defects} * 100 \right] \%$$

It gives the percentage of the invalid defect the testing team has opened and one can control, whenever required.

Defect Rejection Trend



Bad Fix Defect (B)

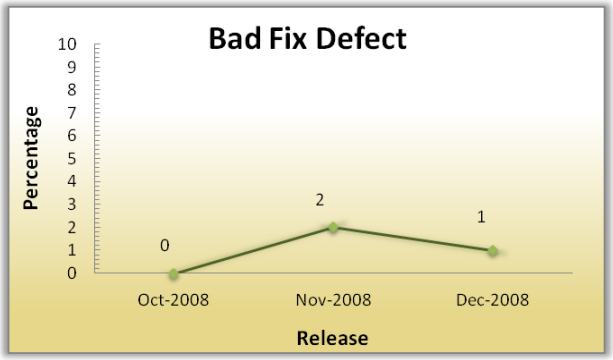
Defect whose resolution give rise to new defect(s) are bad fix defect.

This metric determine the effectiveness of defect resolution process.

$$Bad\ Fix\ Defect = \left[\frac{Number\ of\ of\ Bad\ Fix\ Defect(s)}{Total\ Number\ of\ Valid\ Defects} * 100 \right] \%$$

It gives the percentage of the bad defect resolution which needs to be controlled.

Bad Fix Defect Trend



Test Execution Productivity (TEP)

This metric gives the test cases execution productivity which on further analysis can give conclusive result.

$$Test\ Execution\ Productivity = \left[\frac{Total\ No.\ of\ TC\ executed\ (Te)}{Execution\ Efforts\ (hours)} * 8 \right] Execution(s)/Day$$

Where Te is calculated as,

$$Te = Base\ Test\ Case + ((T\ (0.33) * 0.33) + (T\ (0.66) * 0.66) + (T\ (1) * 1))$$

Where,

Base Test Case = No. of TC executed atleast once.

T (1) = No. of TC Retested with 71% to 100% of Total TC steps

T (0.66) = No. of TC Retested with 41% to 70% of Total TC steps

T (0.33) = No. of TC Retested with 1% to 40% of Total TC steps

Example

TC Name	Base Run Effort (hr)	Re-Run1 Status	Re-Run1 Effort (hr)	Re-Run2 Status	Re-Run2 Effort (hr)	Re-Run3 Status	Re-Run3 Effort (hr)
XYZ_1	2	T(0.66)	1	T(0.66)	0.45	T(1)	2
XYZ_2	1.3	T(0.33)	0.3	T(1)	2		
XYZ_3	2.3	T(1)	1.2				

XYZ_4	2	T(1)	2				
XYZ_5	2.15						

In above example,

Base Test Case	5
T(1)	4
T(0.66)	2
T(0.33)	1
Total Efforts(hr)	19.7

Te = 5 + ((1*4) + (2*0.66) + (1*0.33))) = 5 + 5.65 = 10.65

Test Execution Productivity = (10.65/19.7) * 8 = 4.3 Execution/day

One can compare the productivity with previous release and can have an effective conclusion.

Test Execution Productivity Trend

Test Efficiency (TE)

This metric determine the efficiency of the testing team in identifying the defects.

It also indicated the defects missed out during testing phase which migrated to the next phase.

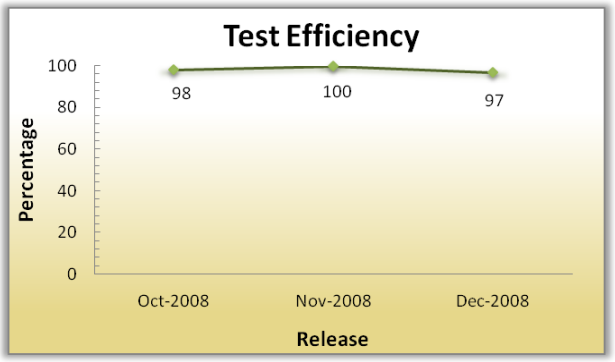
$$Test\ Efficiency = \left[\frac{DT}{DT + DU} * 100 \right] \%$$

Where,

DT = Number of valid defects identified during testing.

DU = Number of valid defects identified by user after release of application. In other words, post-testing defect

Test Efficiency Trend



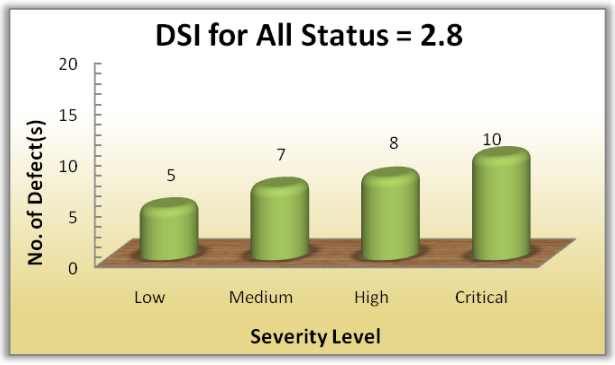
Defect Severity Index (DSI)

This metric determine the quality of the product under test and at the time of release, based on which one can take decision for releasing of the product i.e. it indicates the product quality.

$$Defect\ Severity\ Index = \left[\frac{\sum (Severity\ Index * No.\ of\ Valid\ Defect(s)\ for\ this\ severity}{Total\ Number\ of\ Valid\ Defects} \right]$$

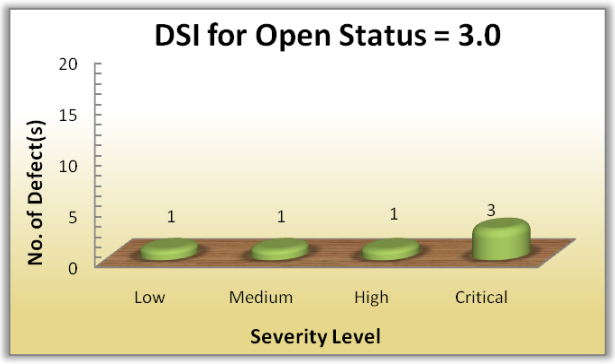
One can divide the Defect Severity Index in two parts: -

1. **DSI for All Status defect(s):** - This value gives the product quality under test.

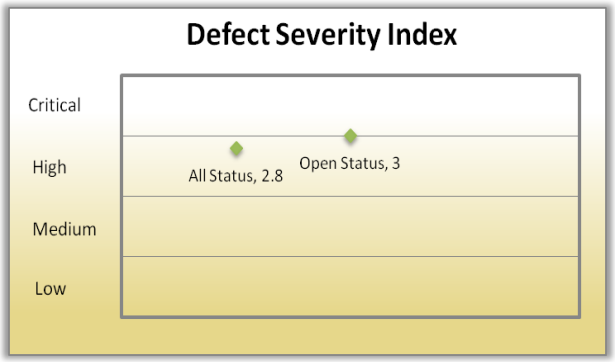


2. **DSI for Open Status defect(s):** - This value gives the product quality at the time of release. For calculation of DSI for this, only open status defect(s) must be considered.

$$DSI(Open) = \left[\frac{\sum (Severity\ Index * No.\ of\ Open\ Valid\ Defect(s)\ for\ this\ severity)}{Total\ Number\ of\ Open\ Valid\ Defects} \right]$$



Defect Severity Index Trend



From the graph it is clear that

- Quality of product under test i.e. DSI – All Status = 2.8 (High Severity)
- Quality of product at the time of release i.e. DSI – Open Status = 3.0 (High Severity)

7.2. Performance Testing Metrics

Performance Scripting Productivity (PSP)

This metric gives the scripting productivity for performance test script and have trend over a period of time.

$$Performance\ Scripting\ Productivity = \left[\frac{\sum Operations\ Performed}{Efforts\ (hours)}\right] Operation(s)/hour$$

Where Operations performed is: -

1. No. of Click(s) i.e. click(s) on which data is refreshed.
2. No. of Input parameter
3. No. of Correlation parameter

Above evaluation process does include logic embedded into the script which is rarely used.

Example

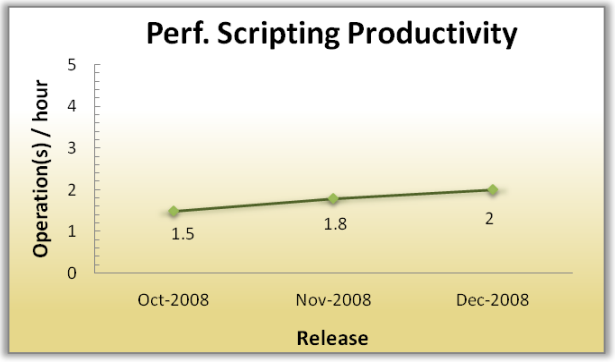
Operation Performed	Total
No. of clicks	10

No. of Input Parameter	5
No. of Correlation Parameter	5
Total Operation Performed	**20**

Efforts took for scripting = 10 hours.

Performance scripting productivity =20/10=2 operations/hour

Performance Scripting Productivity Trend



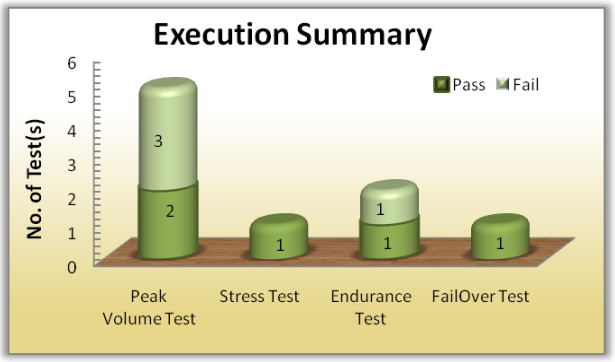
Performance Execution Summary

This metric gives classification with respect to number of test conducted along with status (Pass/Fail), for various types of performance testing.

Some of the types of performance testing: -

1. Peak Volume Test.
2. Endurance/Soak Test.
3. Breakpoint/Stress Test.
4. Failover Test

Summary Trend



Performance Execution Data - Client Side

This metric gives the detail information of Client side data for execution.

Following are some of the data points of this metric -

1. Running Users
2. Response Time
3. Hits per Second
4. Throughput
5. Total Transaction per second
6. Time to first byte
7. Error per second

Performance Execution Data - Server Side

This metric gives the detail information of Server side date for execution.

Following are some of the data points of this metric -

1. CPU Utilization
2. Memory Utilization
3. HEAP Memory Utilization
4. Database connections per second

Performance Test Efficiency (PTE)

This metric determine the quality of the Performance testing team in meeting the requirements which can be used as an input for further improvisation, if required.

$$Performance\,Test\,Efficiency = \left[\frac{(Req\ met\ during\ PT) - (Req\ not\ met\ after\ Signoff\ of\ PT)}{Req\ met\ during\ PT} * 100 \right] \%$$

To evaluate this one need to collect data point during the performance testing and after the signoff of the performance testing.

Some of the requirements of Performance testing are: -

1. Average response time.
2. Transaction per Second.
3. Application must be able to handle predefined max user load.
4. Server Stability.

Example

Consider during the performance testing above mentioned requirements were met.

Requirement met during PT = 4

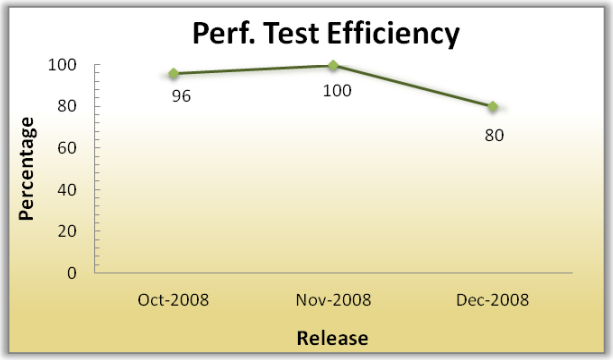
In production, average response time is greater than expected, then

Requirement not met after Signoff of PT = 1

PTE = (4 / (4+1)) * 100 = 80%

Performance Testing Efficiency is 80%

Performance Test Efficiency Trend



Performance Severity Index (PSI)

This metric determine the product quality based performance criteria on which one can take decision for releasing of the product to next phase i.e. it indicates quality of product under test with respect to performance.

$$Performance\ Severity\ Index = \left[\frac{\sum (Severity\ Index * No.\ of\ \ Req.\ not\ met\ \ for\ this\ severity)}{Total\ No.\ of\ Req.\ not\ met} \right]$$

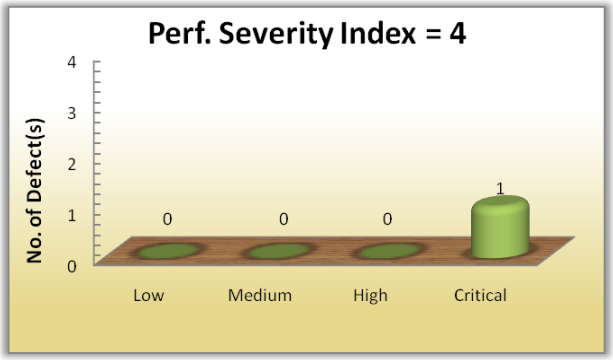
If requirement is not met, one can assign the severity for the requirement so that decision can be taken for the product release with respect to performance.

Example

Consider, Average response time is important requirement which has not met, then tester can open defect with Severity as Critical.

Then Performance Severity Index = (4 * 1) / 1 = 4 (Critical)

Performance Severity Trend



7.3. Automation Testing Metrics

Automation Scripting Productivity (ASP)

This metric gives the scripting productivity for automation test script based on which one can analyze and draw most effective conclusion from the same.

$$Automation\ Scripting\ Productivity = \left[\frac{\sum Operations\ Performed}{Efforts\ (hours)}\right] Operation(s)/hour$$

Where Operations performed is: -

1. No. of Click(s) i.e. click(s) on which data is refreshed.
2. No. of Input parameter
3. No. of Checkpoint added

Above process does include logic embedded into the script which is rarely used.

Example

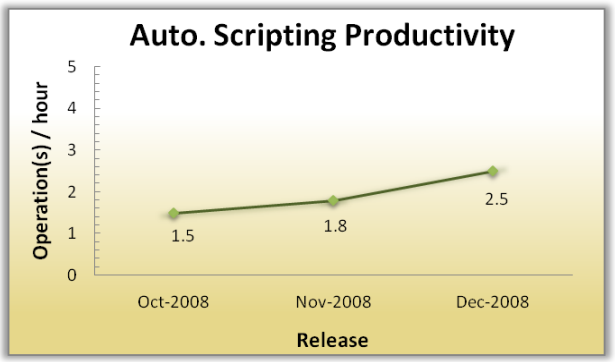
Operation Performed	Total
No. of clicks	10
No. of Input Parameter	5
No. of Checkpoint added	10
Total Operation Performed	**25**

Efforts took for scripting = 10 hours.

ASP=25/10=2.5

Automation scripting productivity = 2.5 operations/hour

Automation Scripting Productivity Trend



Automation Test Execution Productivity (AEP)

This metric gives the automated test case execution productivity.

$$Auto.\ Execution\ Productivity = \left[\frac{Total\ No.\ of\ Automated\ TC\ executed\ (ATe)}{Execution\ Efforts\ (hours)} * 8 \right] Execution(s)/Day$$

Where ATe is calculated as,

$$ATe = Base\ Test\ Case + ((T\ (0.33) * 0.33) + (T\ (0.66) * 0.66) + (T\ (1) * 1))$$

Evaluation process is similar to Manual Test Execution Productivity.

Automation Coverage

This metric gives the percentage of manual test cases automated.

$$Automation\ Coverage = \left[\frac{Total\ No.\ of\ TC\ Automated}{Total\ No.\ of\ Manual\ TC} * 100 \right] \%$$

Example

If there are 100 Manual test cases and one has automated 60 test cases then Automation Coverage = 60%

Cost Comparison

This metrics gives the cost comparison between manual testing and automation testing. This metrics is used to have conclusive ROI (return on investment).

Manual Cost is evaluated as: -

Cost (M) = Execution Efforts (hours) * Billing Rate

Automation cost is evaluated as: -

Cost (A) = Tool Purchased Cost (One time investment) + Maintenance Cost
+ Script Development Cost
+ (Execution Efforts (hrs) * Billing Rate)

If Script is re-used the script development cost will be the script update cost.

Using this metric one can have an effective conclusion with respect to the currency which plays a vital role in IT industry.

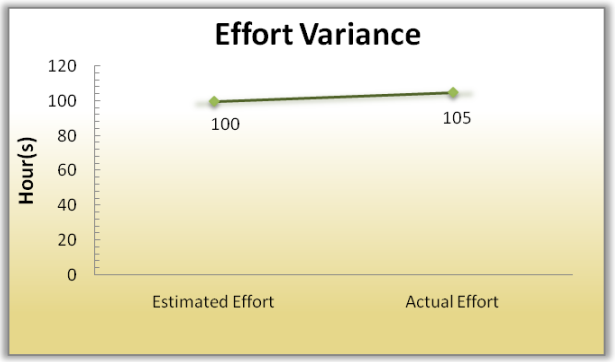
7.4. Common Metrics for all types of testing

Effort Variance (EV)

This metric gives the variance in the estimated effort.

$$Effort\ Variance = \left[\frac{Actual\ Effort - Estimated\ Effort}{Estimated\ Effort} * 100 \right] \%$$

Effort Variance Trend

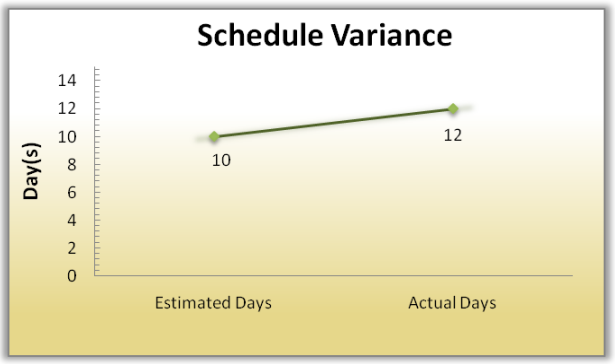


Schedule Variance (SV)

This metric gives the variance in the estimated schedule i.e. number of days.

$$Schedule\ Variance = \left[\frac{Actual\ No.\ of\ Days - Estimated\ No.\ of\ Days}{Estimated\ No.\ of\ Days} * 100 \right] \%$$

Schedule Variance Trend



Scope Change (SC)

This metric indicates how stable the scope of testing is.

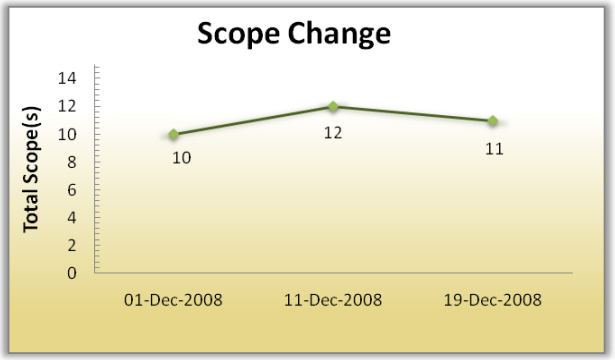
$$Scope\ Change = \left[\frac{Total\ Scope - Previous\ Scope}{Previous\ Scope} * 100 \right] \%$$

Where,

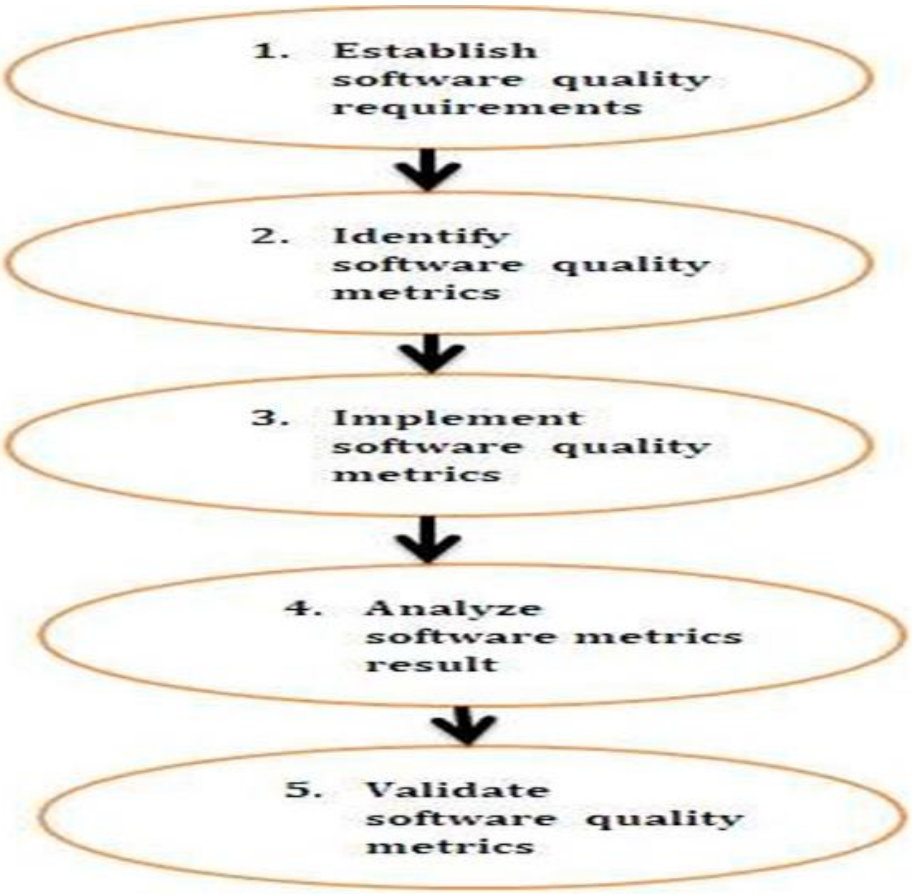
Total Scope = Previous Scope + New Scope, if Scope increases

Total Scope = Previous Scope - New Scope, if Scope decreases

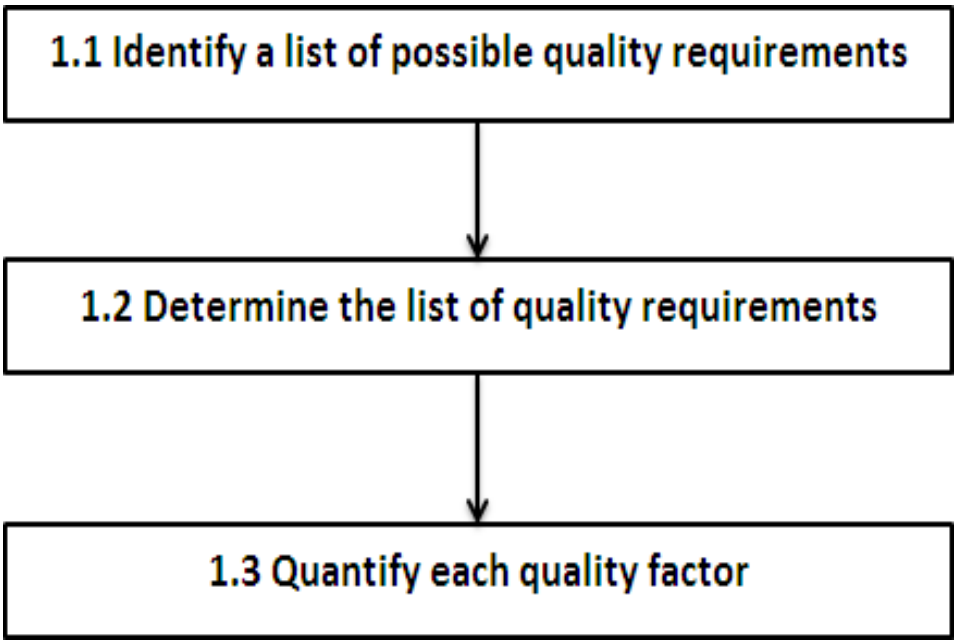
Scope Change Trend for one release



8.Software Metrics Methodology



1. **Establish software quality requirements**



1.1. Identify a list of possible quality requirements

Use,

- organizational experience
- required standards
- regulations
- laws

consider,

- contractual requirements
- cost or schedule constraints
- warranties
- customer metrics requirements
- organizational self-interest

1.2 Determine the list of quality requirements:

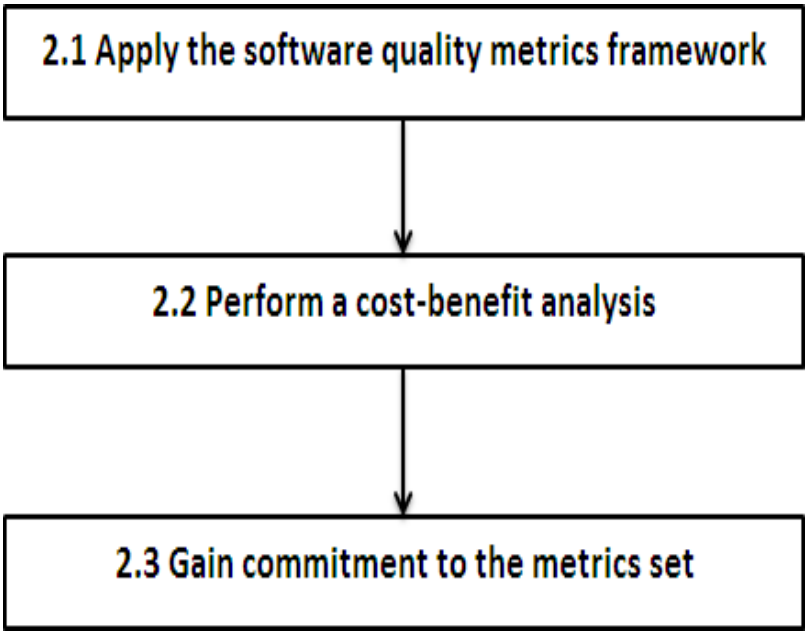
- Survey all involved parties

- Create the list of quality requirements

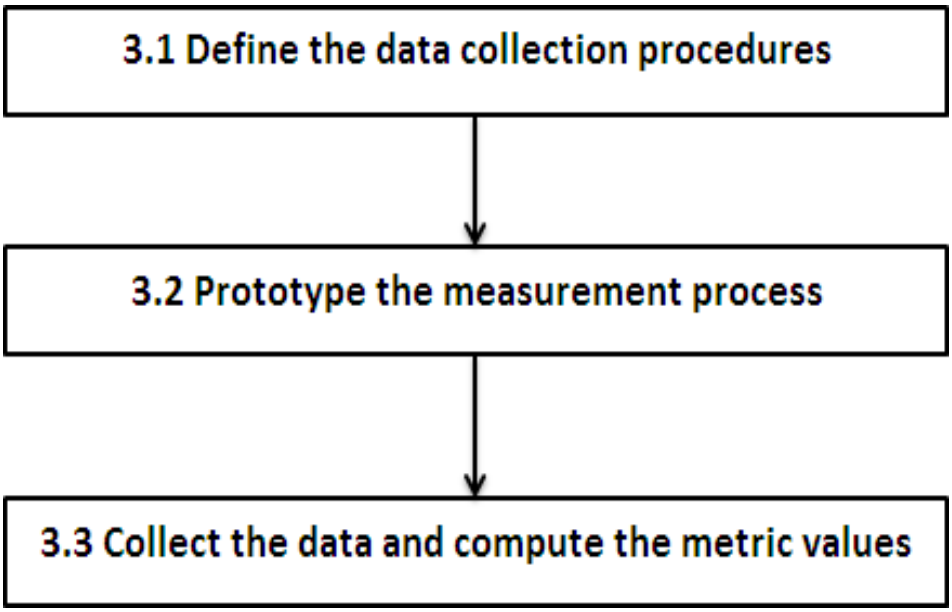
1.3 Quantify each quality factor:

- For each quality factor, assign one or more direct metrics to represent the quality factor
- Assign direct metric values to serve as quantitative requirements for that quality factor

2. IDENTIFY SOFTWARE QUALITY METRICS



3. IMPLEMENT SOFTWARE QUALITY METRICS



- Identify tools
- Describe data storage procedures
- Establish a traceability matrix
- Identify the organizational entities
- Participate in data collection
- Responsible for monitoring data collection
- Describe the training and experience required for data collection
- Training process for personnel involved

Item	Description
Name	Name given to the data item.
Metrics	Metrics that are associated with the data item.
Definition	Straightforward description of the data item.
Source	Location of where the data item originates.
Collector	Entity responsible for collecting the data.
Timing	Time(s) in life cycle at which the data item is to be collected. (Some data items are collected more than once.)
Procedures	Methodology (e.g., automated or manual) used to collect the data.
Storage	Location of where the data are stored.
Representation	Manner in which the data are represented, e.g., precision and format (Boolean, dimensionless, etc.).
Sample	Method used to select the data to be collected and the percentage of the available data that is to be collected.
Verification	Manner in which the collected data are to be checked for errors.
Alternatives	Methods that may be used to collect the data other than the preferred method.
Integrity	Person(s) or organization(s) authorized to alter the data item and under what conditions.

data item description

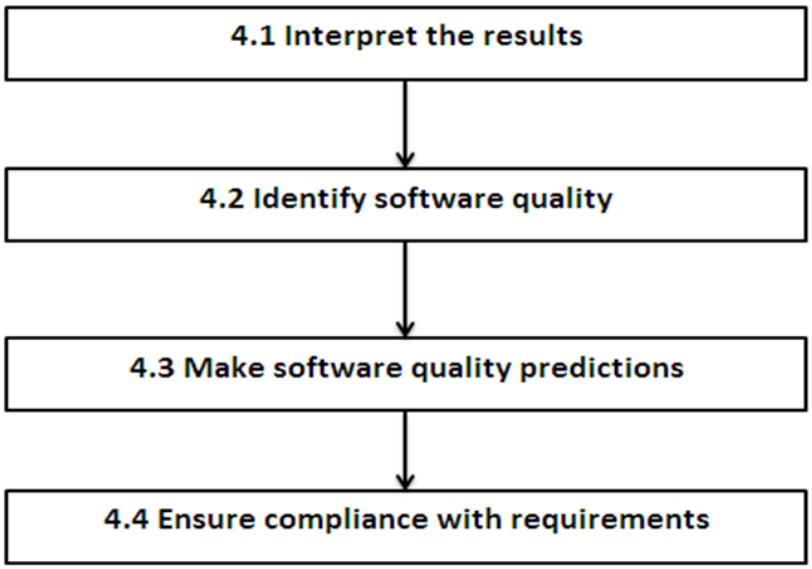
3.1. Define data collection procedures

3.2 prototype the measurement process

- Test the data collection and metrics computation procedures on selected software that will act as a prototype
- Select samples that are similar to the project(s) on which the metrics will be used
- Examine the cost of the measurement process for the prototype to verify or improve the cost analysis
- Results collected from the prototype to improve the metric descriptions and descriptions of data items.
- Using the formats in Table, **collect and store data** in the project metrics database at the appropriate time in the life cycle
- Check the data for **accuracy** and proper unit of measure
- **Monitor** the data collection

- Check for **uniformity** of data if more than one person is collecting it
- **Compute** the metric values from the collected data

4. analyse the software metric result



4.1 Interpret the results

- The purpose of metrics validation is to identify both product and process metrics that can predict specified quality factor values, which are quantitative representations of quality requirements
- Metrics shall indicate whether quality requirements have been achieved or are likely to be achieved in the future

- For the purpose of assessing whether a metric is valid
- The following thresholds shall be designated:

V-square of the linear correlation coefficient

B-rank correlation coefficient

A-prediction error @-confidence level P-success rate

- Correlation
- Tracking
- Consistency
- Predictability
- Discriminative power
- Reliability

5. VALIDATE THE SOFTARE QUALITY

5.3.1 Identify the quality factors sample

 A sample of quality factors shall be drawn from the metrics database

5.3.2 Identify the metrics sample

 A sample from the same domain (e.g., same software components), as used in 5.3.1, shall be drawn from the metrics database

5.3.3 Perform a statistical analysis

 The analysis described in 5.2 shall be performed

 Before a metric is used to evaluate the quality of a product or process, it shall be validated against the criteria described in

If a metric does not pass all of the validity tests, it shall only be used according to the criteria prescribed by those tests

5.3.4 Document the results

 Documented results shall include the direct metric, predictive metric, validation criteria, and numerical results, as a minimum

5.3.5 Revalidate the metrics

A validated metric may not necessarily be valid in other environments or future applications. Therefore, a predictive metric shall be revalidated before it is used for another environment or application

5.3.6 Evaluate the stability of the environment

Metrics validation shall be undertaken in a stable development environment (i.e., where the design language, implementation language, or project development tools do not change over the life of the project in which validation is performed)

9. Software quality indicator

A Software Quality Indicator is used to calculate and to provide an indication of the quality of the system by assessing system characteristics. The software quality indicators address management concerns. It is also used for decision making by decision maker. It includes a measure of the reliability of the code.

An indicator usually compares a metric with a baseline or expected result.

It acts as a set of tools to improve the management capabilities of personnel responsible for monitoring

Software quality indicators extract from requirements are flexibility and adaptability.

Following quality indicators can use during the software testing & development life cycle.

1] Progress:-

It Measures the amount of work accomplished by the developer in each phase

2] Stability:-

Assesses whether the products of each phase are sufficiently stable to allow the next phase to proceed

3] Process compliance:-

It compliance with the development procedures approved at the beginning of the project to the running development process

4] Quality Evaluation efforts:-

It evaluates percentage of the developer's effort that is being spent on internal quality evaluation activities and time required to deal.

5] Test coverage:-

It measures the amount of the software system / functionality covered by the developer's testing process.

6] Defect detection efficiency:-

Measures how many of the defects detectable discovered during that phase.

7] Defect removal rate:-

Total number of defects detected and resolved over a period of time

8] Defect density:-

Detects defect-prone components of the system

9]Defectageprofile:-

It measures the number of defects that have remained unresolved for a long period of time.

10]Complexity:-

It measures the complexity of the code. It counts the total path, branch, coverage to calculate the complexity

10. Fundamentals of Measurement Theory

It is undisputed that measurement is crucial to the progress of all sciences. Scientific progress is made through observations and generalizations based on data and measurements, the derivation of theories as a result and in turn the confirmation or refutation of theories via hypothesis testing based on further empirical data. As an example, consider the proposition "the more rigorously the front end of the software development process is executed, the better the quality at the back end." To confirm or software development process" and distinguish the process step and activities of the front end from those of the back end. Assume that after the requirements gathering process, our development process consists of the following phases:

• Design

• Design reviews and inspections
• Code
• Code inspection
• Debug and development tests
• Integration of components and modules to form the product
• Formal machine testing
• Early customer programs

Integration is the development phase during which various parts and components are integrated to form one complete software product. Usually after integration the product is under formal change control. Specifically, after integration every change of the software must have a specific reason (e.g., to fix a bug uncovered during testing) and must be documented and tracked. Therefore, we may want to use integration as the cutoff point: The design, coding, debugging, and integration phase are classified as the front end of the development process and the formal machine testing and early customer trials constitute the back end.

We then define rigorous implementation both in the general sense and in specific terms as they relate to the front end of the development process. Assuming the development process has been formally documented, we may define rigorous implementation as total adherence to the process: whatever is described in the process documentation that needs to be executed, we execute. However, this general on is not sufficient for our purpose, which is to gather data to test our propositions. We need to specify the indicator(s) of the definition and to make it (them) operational. For example, suppose the process documentation says all designs and code should be inspected. One operational definition of rigorous implementation may be inspection coverage expressed in terms of the percentage of the estimated lines of code (LOC) or of thee function [points (FP) that are actually inspected. Another indicator of good reviews and inspections could be the scoring of ach inspections by the inspectors at the end of the inspection, based on a set of criteria. We may want to operationally use a five – point Likert scale to denote the degree of effectiveness (e.g., 5 = very effective, 4 = effective, 3 = somewhat effective,2 = not effective, 1 = poor inspection). There may also be other indicators.

In addition to design, design reviews, code implementation, and code inspections, development testing is part of our definition of the front end of the development process. We also need to operationally define “rigorous execution“of this test. Two indicators that could be used are the

percent coverage in terms of instructions executed (as measured by some test coverage measurement tools) and the defect rate expressed in terms of number of defects removed per thousand lines of source code (KLOC) or per function point. Likewise, we need to operationally define "quality at the back end" and decide which measurement indicators to us. For the sake of simplicity let us use defects found per KLOC (or defects per function point) during formal machine testing as the indicator of back end quality. From these metrics, we can formulate several testable hypotheses such as the following:

• For software projects, the higher the percentage of the design and code that are inspected, the lower the defect rat at the later phase of formal machine testing.

• The more effective the design reviews and the code inspections as scored by the inspection team, the lower the defect rate at the later phase of formal machine testing.

• The more through the development testing (in terms of test coverage) before integration, the lower the defect rate at the formal machine testing phase.

With the hypotheses formulated, we can set out to gather data and test the hypotheses. We also need to determine the unit of analysis for our measurement and data. In this case, it could be at the project level or at the component level of a large project. If we are able to collect a number of data points that from a reasonable sample size (e.g., 35 projects or components), we can perform statistical analysis to test the hypotheses. We can classify projects or components into several groups according to the independent variable of each hypothesis, and then compare the outcome of the dependent variable (defect rate during formal machine testing) across the groups. We can conduct simple correlation analysis, or we can perform more sophisticated statistical analyses. If the hypotheses are substantiated by the data, we confirm the proposition. If they are rejected, we refute the proposition. If we have doubts or unanswered questions during the process (e.g., are our indicators valid? Are our data reliable? Are there other variables we need to control when we conduct the analysis for hypothesis testing?, then perhaps more research is needed. However, if the hypothesis (e s) or the proposition is confirmed, we can use the knowledge thus gained and act accordingly to improve our software development quality.

11. LEVEL OF MEASUREMENT

We have seen that from theory to empirical hypothesis and from theoretically defined concepts to operational definitions, the process is by no means direct .As the example illustrates, when we operationalize a definition and derive measurement indicators, we must consider the scale of measurement. For instance, to measure the quality of software inspection we may use a five-point scale to score the inspection effectiveness or we may use percentage to indicate the inspection coverage. For some cases, more than one measurement scale is applicable; for others, the nature of the concept and the resultant operational definition can b measured only with a certain scale. In this section, we briefly discuss the four levels of measurement: normal scale, ordinal scale, interval scale, and ratio scale.

11.1. Nominal Scale

The most simple operation in science and the lowest level of measurement is classification. In classifying we attempt to sort elements into categories with respect to a certain attribute. For example, if the attribute of interest is religion, we may classify the subjects of the study into Catholics, Protestants, Jews, Buddhists, and so on. If we classify software products by the development process models through which the products were developed, then we may have categories such as waterfall development process and other. In a normal scale, the two key requirements for the categories are jointly exhaustive and mutually exclusive. Mutually exclusive mans a subject can be classified into one and only one category. Jointly exhaustive means that all categories together should cover all possible categories of the attribute. If the attribute has more categories than we are interested in, an “other” category is needed to make the categories jointly exhaustive.

In a normal scale, the names of the categories and their sequence bear no assumptions about relationships among categories. For instance, we place the waterfall development process in front of spiral development process, but we do not imply that one is “better than” or “greater than” the other. As long as the requirements of mutually exclusive and jointly exhaustive are met, we may want to compare the values of interested attributes such as defect rate, cycle time, and requirements defects across the different categories of software products.

11.2. Ordinal Scale

Ordinal scale refers to the measurement operations through which the subjects can be compared in order. For example, we may classify families according to socioeconomic status: upper class, middle class, and lower class. We may classify software development projects according to the SEI maturity levels or according to a process rigor scale: totally adheres to process, somewhat adheres to process, does not adhere to process. Our earlier example of inspection effectiveness scoring is an ordinal scale.

The ordinal measurement scale is at a higher level than the nominal scale in the measurement hierarchy. Through it we are able not only to group subjects into categories, but also to order the categories. An ordinal scale is asymmetric in the sense that if A>B is true then B>A is false. It has the transitivity property in that if A>B and B>C, A>C.

We must recognize that an ordinal scale offers no information about the magnitude of the differences between elements. For instance, for the process rigor scale we know only that "totally adheres to process" is better than "somewhat adheres to process" in terms of the quality outcome of the software product, and "somewhat adheres to process" is better than "dos not adheres to process." However, we cannot say that the difference between the former pair of categories is the same as that between the letter pair. In customer satisfaction surveys of software products, the five-point Likert scale is often used with 1=completely dissatisfied, 2= somewhat dissatisfied, 3= neutral, 4= satisfied, and 5= completely satisfied. We know only 5>4, 4>3, and 5>2, and so forth, but we cannot say how much greater 5 is than4. Nor can we say that the difference between categories 5and 4 is equal to that between categories 3 and 2, Indeed, to move customers from satisfied (4) to very satisfied (5) versus from dissatisfied (2) to neutral (3) , to neutral (3), may require very different actions and types of improvements.

Therefore, when we translate order relations into mathematical operations, we cannot use operations such as addition, subtraction, multiplication, and division. We can use "greater than" and "less than". However, in real-world application for some specific types of ordinal scales (such as the Likert five-point, seven-point, or ten-point scales), the assumption of equal distance is often made and operations such as averaging are applied to these scales. In such cases, we should be aware that the

measurement assumption is deviated, and then use extreme caution when interpreting the results of data analysis.

11.3. Interval and Ratio Scales

An interval scale indicates the extract differences between measurement points. The mathematical operations of addition and substraction can be applied to interval scale data .For instance, assuming products A, B, and C are developed in the same language, if the defect rate of software product A is 5 defects per KLOC and product B's rate is 3.5 defects per KLOC, then we can say product A's defect level is 1.5 defects per KLOC higher than product B's defect level. An interval scale of measurement requires a well-defined unit of measurement that can be agreed on as a common standard and that is repeatable. Given a unit of measurement, it is possible to say that the difference between two scores is 15 units or that one difference is the same as a second. Assuming product C's defect rate is 2 defects per KLOC, we can thus say the difference in defect rate between products A and B is the same as that between B and C.

When an absolute or no arbitrary zero point can be located on an interval scale, it becomes a ratio scale. Ratio scale is the highest level of measurement and all mathematical operations can be applied to it, including division and multiplication. For example, we can say that product A's defect rate is twice as much as product C's because when the defect rate is zero, that means not a single defects exists in the product. Had the zero point been arbitrary, the statement would have been illegitimate. A good example of an interval scale with an arbitrary zero point is the traditional temperature measurement (Fahrenheit and centigrade scales).Thus we say that the difference between the average summer temperature (80 degree F) and the average winter temperature (16 degree F is 64 degree F, but we do not say that 80 degree F is five times as hot as 16 degree F Fahrenheit and centigrade temperature scales are interval, not ratio, scales. For this reason, scientists developed the absolute temperature scale (a ratio scale) for use in scientific activities.

Except for a few notable examples, for all practical purposes almost all interval measurement scales are also ratio scales. When the size of the unit is established, it is usually possible to conceive of a zero unit.

For internal and ratio scales, the measurement can be expressed in both integer and no integer data.

Integer data are usually given in terms of frequency counts (e.g., the number of defects customers will encounter for a software product over a specified time length).

We should note that the measurement scales are hierarchical. Each higher level scale possesses all properties of the lower ones. The higher the level of measurement, the more powerful analysis can be applied to the data. Therefore, in our operationalization process we should devise metrics that can take advantage of the highest level of measurement allowed by the nature of the concept and its definition. A higher-level measurement can always make various types of comparisons if the scale is in terms of actual defect rate. However, if the scale is in terms of excellent, good, average, worse than average, and poor, as compared to an industrial standard, then our ability to perform additional analysis of the data is limited.

12. SOME BASIC MEASURES

Regardless of the measurement scale, when the data are gathered we need to analyze them to extract meaningful information. Various measures and statistics are available for summarizing the raw data and for making comparisons across groups. In this section we discuss some basic measures such as ratio, proportion, percentage, and rate, which are frequently used in our daily lives as well as in various activities associated with software development and software quality. These basic measures, while seemingly easy, are often misused. There are also numerous sophisticated statistical techniques and methodologies that can be employed in data analysis. However, such topics are not within the scope of this discussion.

12.1.Ratio

A ratio results from dividing one quality by another. The numerator and denominator are from two distinct populations and are mutually exclusive. For example, in demography, sex ratio is defined as:

Number of males
---------- X 100 %
Number of females

If the ratio is less than 100, there are more females than males: otherwise there are more males than females.

Ratios are also used in software metrics. The most often used, perhaps, is the ratio of number of people in an independent test organization to the number of those in the development group. The test/ development head-count ratio could range from 1:1 to 1:10 depending on the management approach to the software development process. For the large-ratio (e.g., 1:10) organizations, the development group usually is responsible for the complete development (including extensive development tests) of the product, and the test group conducts system-level testing in terms of customer environment verifications. For the small-ratio organizations, the independent group takes the major responsibility for testing (after debugging and code integration) and quality assurance.

11.2. Proportion

Proportion is different from ratio in that the numerator in a proportion is a part of the denominator:

$P = a/a + b$

Proportion also differs from ratio in that the ratio is best used for two groups, whereas proportion is used for multiple categories (or populations) of one group. In other words, the denominator in the preceding formula can be more than just a+b. If

$a + b + c + d + e + N$

Then we have

$a/N + b/N + c/N + d/N + e/N = 1$

When the numerator and the denominator are integers and represent counts of certain events, then p is also referred to as a relative frequency. For example, the following gives the proportion of satisfied customers of the total customer set:

Number of satisfied customers

Total number of customers of a software product

The numerator and the denominator in a proportion need not be integers. Thy can be frequency counts as well as measurement units on a continuous scale (e.g., height in inches, weight in pounds).When the measurement unit is not integer, proportions are called fractions.

11.3.Percentage

A proportion or a fraction becomes a percentage when it is expressed in terms of per hundred units (the denominator is normalized to 100). The word percent mans per hundred. A proportion p is therefore equal to 100%; percent (100/7%).

Percentages are frequently used to report results, and as such are frequently misused. First, because percentages represent relative frequencies, it is important that enough contextual information be given, specially the total number of cases, so that the readers can interpret the information correctly. Jones (1992) observes that many reports and presentations in the software industry are careless in using percentages and ratios. He cites the example:

Requirements bugs were 15% of the total, design bugs were 25% of the total, coding bugs were 50% of the total, and other bugs made up 10% of the total.

Had the results been stated as follows, it would have been much more informative:

The project consists of 8 thousand lines of code (KLOC). During its development a total of 200 defects were detected and removed, giving a defect removal rate of 25 defects per KLOC. Of the 200 defects, requirements bugs constituted 15%, design bugs 25%, coding bugs 50%, and other bugs made up 10%.

A second important rule of thumb is that the total number of cases must be sufficiently large enough to use percentages. Percentages computed from a small total are not stable: they also convey an impression that a large number of cases are involved. Some writers recommend that the minimum number of cases for which percentages should b calculated is 50. We recommend that, depending on the number of categories, the minimum number be 30, the smallest sample size required for parametric statistics. If the number of cases is too small, then absolute numbers, instead of percentages, should be used. For instance,

Of the total 20 defects for the entire project of 2 KLOC, there were 3 requirements bugs, 5 design bugs, 10 coding bugs, and 2 others.

When results in percentages appear in table format, usually both the percentages and actual numbers are shown when there is only one variable. When there are more than two groups, such as the example in Table 1, it is better just to show the percentages and the total number of cases (N) for each group. With percentages and N known, one can always reconstruct the frequency distributions. The total of 100.0% should always be shown so that it is clear how the percentages are computed. In a two-way table, the direction in which the percentages are computed depends on the purpose of the comparison. For instance, the percentages in Table 1 are computed vertically (the total of each column is 100.0%), and the purpose is to compare the defect type- profile across projects (e.g., project B proportionally has more requirements defects than project A).

In Table 2, the percentages are computed horizontally. The purpose here is to compare the distribution of defects across projects for each type of defect. The interpretations of the two tables differ. Therefore, it is important to carefully examine percentage tables to determine exactly how the percentages are calculated.

School of Computing

Department of Computer Science and Engineering

and

Department of Information Technology

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

UNIT III

Software Testing Strategies and Approaches

1. Static Testing Strategy

A static test evaluates the quality of a system without actually running the system. While that may seem impossible, it can be accomplished in a few ways.

The static test looks at portions of or elements related to the system in order to detect problems as early as possible. For example, developers review their code after writing and before pushing it. This is called desk-checking, a form of static testing. Another example of a static test would be a review meeting held for the purpose of evaluating requirements, design, and code.

Static tests offer a decided advantage: If a problem is detected in the requirements before it develops into a bug in the system, it will save time and money. If a preliminary code review leads to bug detection, it saves the trouble of building, installing, and running a system to find and fix the bug.

It is possible to perform automated static tests with the right tools. C programmers can use the lint program to identify potential bugs while Java users can utilize JTest to check their scripts against a coding standard.

Static tests must be performed at the right time. For example, if requirements are reviewed after developers have finished coding the entire software it can help testers design test cases. But testers cannot detect bugs in already written code without running the system, thus defeating the purpose of static tests. In this case, the code must be reviewed by individual developers as soon as it is created, and before it is integrated.

Additionally, static tests must be run not just by technical personnel, but other stakeholders. Business domain experts must review requirements, system architects must review design, and so on. Testers' feedback is also imperative since they are trained to spot inconsistencies, missing details, vague functionality, etc.

2. Structural Testing Strategy

While static tests are quite useful, they are not adequate. The software needs to be operated on real devices, and the system has to be run in its entirety to find all bugs. Structural tests are among the most important of these tests.

Structural tests are designed on the basis of the software structure. They can also be called white-box tests because they are run by testers with thorough knowledge of the software as well as the devices and systems it is functioning on. Structural tests are most often run on individual components and interfaces in order to identify localized errors in data flows.

A good example of this would be using reusable, automated test harnesses for the system being tested. With this harness in place, coders can create structural test cases for components right after they have written the code for each component. Then, they register the tests into the source code repository along with the main component during integration. A well-crafted test harness will run the tests every time new code is added, thus serving as a regression test suite.

Since creating structural tests require a thorough understanding of the software being tested, it is best that they are executed by developers or highly skilled testers. In the best-case scenario, developers and testers work in tandem to set up test harnesses and run them at regular intervals. Testers are especially helpful when it comes to developing reusable and shareable test scripts and cases, which cut down on time and effort in the long run.

3. Behavioral Testing Strategy

Behavioral Testing focuses on how a system acts rather than the mechanism behind its functions. It focuses on workflows, configurations, performance, and all elements of the user journey. The point of these tests, often called “black box” tests, is to test a website or app from the perspective of an end-user.

Behavioral testing must cover multiple user profiles as well as usage scenarios. Most of these tests focus on fully integrated systems rather than individual components. This is because it is possible to effectively gauge system behavior from a user’s eyes, only after it has been assembled and integrated to a significant extent.

Behavioral tests are most frequently run manually, though some of them can be automated. Manual testing requires careful planning, design, and meticulous checking of results

to detect what goes wrong. Skilled manual testers are known for being able to follow a trail of bugs and ascertain their effect on user experience.

Automation testing helps primarily to run repetitive actions, such as regression tests which check that new code has not disrupted already existing features that are working well. For example, a website needs to be tested by filling 50 fields in a form. Now this action needs to be repeated with multiple sets of values. Obviously, it is smarter to let a machine handle this rather than risk wasting time, human effort, and human error.

Behavioral testing does require some understanding of the system's technicality. Testers need some measure of insight into the business side of the software, especially with regard to what target users want. In order to plan test scenarios, they must know what users are likely to do once they access a website or app.

What to consider when choosing a software testing strategy?

A strategic approach to software testing must take the following into account:

- **Risks-** Risk management is very important during testing to figure out the risks and the risk level. For example, for an app that is well-established and slowly evolving, regression is a critical risk.
- **Objectives-** Testing should satisfy the requirements and needs of stakeholders to succeed. The objective is to look for as many defects as possible with less up-front time and effort invested.
- **Skills-** It is important to consider the skills of the testers since strategies should not only be chosen but executed as well. A standard-compliant strategy is a smart option when lacking skills and time in the team to create an approach.
- **Product-** Some products have specified requirements. This could lead to synergy with an analytical strategy that is requirements-based.
- **Business-** Business considerations and strategy are often important. If using a legacy system as a model for a new one, you could use a model-based strategy.

- **Regulations-** At some instances, one needs to satisfy the regulators along with the stakeholders. In this case, you would need a methodical strategy which satisfies these regulators.

The role of Real Devices: An Accurate Software Testing Approach

The point of all software testing is to identify bugs. Testers must be perfectly clear on how frequently a bug occurs and how it affects the software.

The best way to detect all bugs is to run software through real devices and browsers. When it comes to a website, ensure that it is under the purview of both manual testing and automation testing. Automated Selenium testing should supplement manual tests so that testers do not miss any bugs in the Quality Assurance process.

Websites must also be put through extensive cross browser testing so that they function consistently, regardless of the browser they are being accessed by. Using browsers installed on real devices is the only way to guarantee cross-browser compatibility and not alienate users of any browser.

The best option is to opt for a cloud-based testing service that provides real device browsers and operating systems. BrowserStack offers **2000+ real browsers and devices** for manual and automated testing. Users can sign up for free, log in, choose desired device-browser-OS combinations and start testing.

The same applies to apps. BrowserStack offers real devices for mobile app testing and automated app testing. Simply upload the app to the required device-OS combination and check to see how it functions in the real world.

Additionally, BrowserStack offers a wide range of debugging tools that make it easy to share and resolve bugs. This includes text and video logs to identify exactly where and why a test failed, thus letting testers zero in on what issue to work on.

A clear comprehension of test automation strategy is essential to building test suites, scripts and timelines that offer fast and accurate results. This is equally true for manual tests. Don't start testing without knowing what techniques to use, what approach to follow and how the software is expected to perform. The information in this article intends to provide a starting point for

building constructive testing plans, by detailing what strategies exist for testers to explore in the first place.

Strategy of testing

A strategy of software testing is shown in the context of spiral.

Following figure shows the testing strategy:

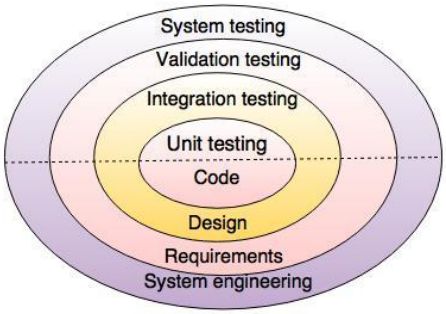


Fig. - Testing Strategy

Unit testing

Unit testing starts at the centre and each unit is implemented in source code.

Integration testing

An integration testing focuses on the construction and design of the software.

Validation testing

Check all the requirements like functional, behavioral and performance requirement are validate against the construction software.

System testing

System testing confirms all system elements and performance are tested entirely.

Testing strategy for procedural point of view

As per the procedural point of view the testing includes following steps.

1) Unit testing
2) Integration testing
3) High-order tests
4) Validation testing

These steps are shown in following figure:

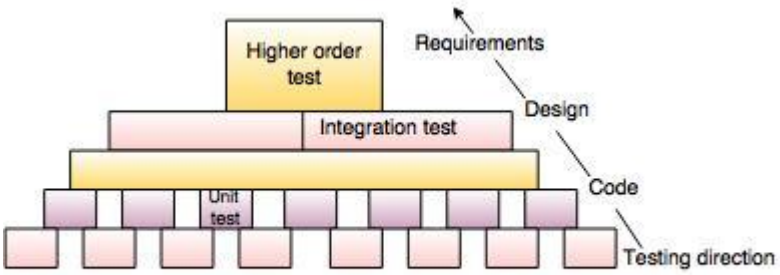
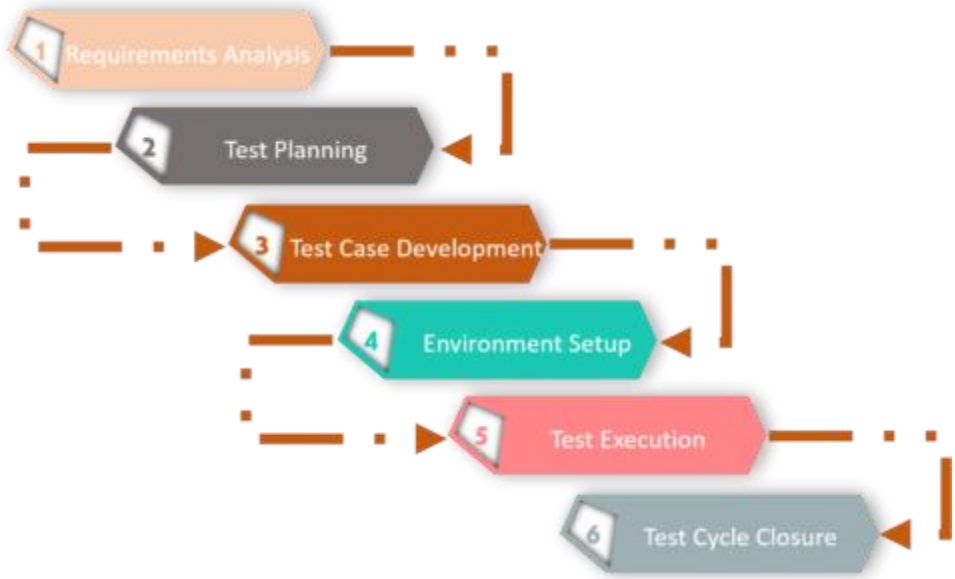


Fig.- Steps of software testing

Software Testing Life Cycle – Different Stages of Testing

What is Software Testing Life Cycle (STLC)?

Software Testing Life Cycle (STLC) defines a series of activities conducted to perform Software Testing. It identifies what test activities to carry out and when to accomplish those test activities. In the STLC process, each activity is carried out in a planned and systematic way and each phase has different goals and deliverables.



What are the different phases of Software Testing Life Cycle?

The different phases of Software testing life cycle are:

- Requirement Analysis
- Test Planning
- Test Case Development
- Environment Setup
- Test Execution
- Test Cycle Closure

Now let's move ahead and have a look at the different phases of software testing life cycle in detail.

Requirement Analysis

Requirement Analysis is the first step involved in Software testing life cycle. In this step, Quality Assurance (QA) team understands the requirement in terms of what we will testing & figure out the testable requirements. During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.The QA team may interact with various stakeholders such as client, business analyst, technical leads, system architects etc. to understand the requirements in detail.

The different types of Requirements include :

Business Requirements – They are high-level requirements that are taken from the business case from the projects.

Architectural & Design Requirements – These requirements are more detailed than business requirements. It determines the overall design required to implement the business requirement.

System & Integration Requirements – It is detailed description of each and every requirement. It can be in form of user stories which is really describing everyday business language. The requirements are in abundant details so that developers can begin coding.

Entry Criteria	Deliverable
The following documents are required: • Requirements Specification. • Application architectural	• List of questions with all answers to be resolved from testable requirements • Automation feasibility report

Activities
• Prepare the list of questions or queries and get resolved from Business Analyst, System Architecture, Client, Technical Manager/Lead etc. • Make out the list for what all Types of Tests performed like Functional, Security, and Performance etc. • Define the testing focus and priorities. • List down the Test environment details where testing activities will be carried out.

- Checkout the Automation feasibility if required & prepare the Automation feasibility report.

Test Planning

Test Planning is the most important phase of Software testing life cycle where all testing strategy is defined. This phase is also called as **Test Strategy** phase. In this phase, the Test Manager is involved to determine the effort and cost estimates for the entire project. It defines the objective & scope of the project.

The commonly used Testing types are :

- Unit Test
- API Testing
- Integration Test
- System Test
- Install/Uninstall Testing
- Agile Testing

Test plan is one of the most important steps in software testing life cycle. The steps involved in writing a test plan include :

1. Analyze the product
2. Design Test Strategy
3. Define Test Objectives
4. Define Test Criteria
5. Resource Planning
6. Plan Test Environment
7. Schedule & Estimation
8. Determine Test Deliverable

Test Case Development

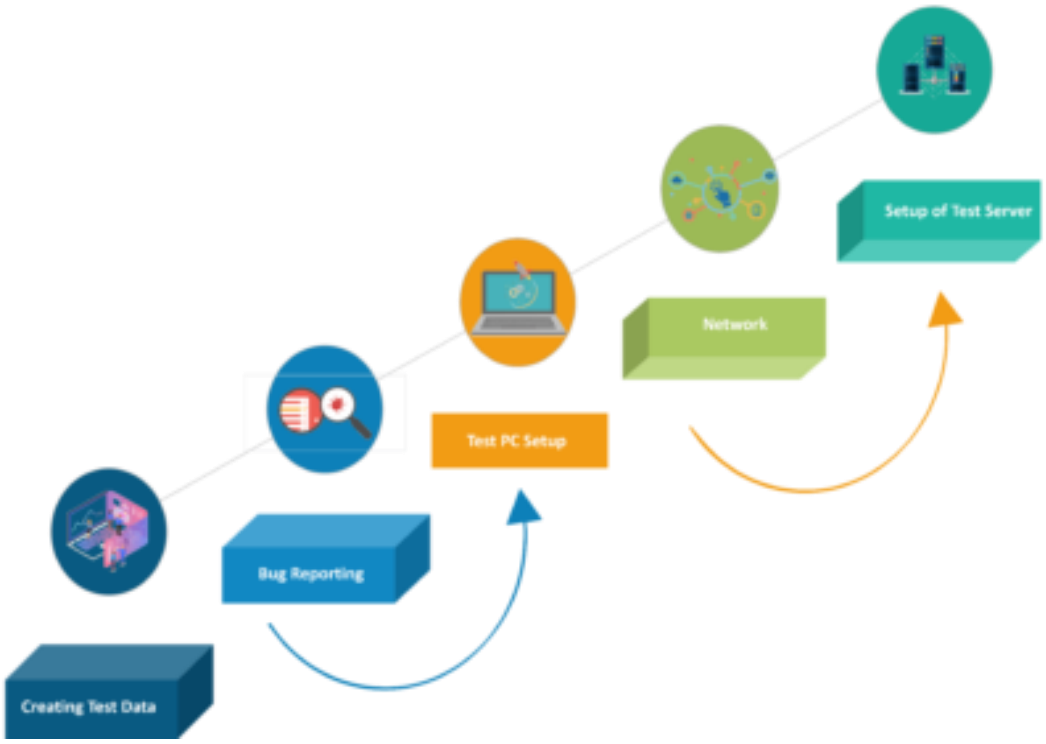
The Test case development begins once the test planning phase is completed. This is the phase of STLC where testing team notes the detailed test cases. Along with test cases, testing team also prepares the test data for testing. Once the test cases are ready then these test cases are reviewed by peer members or QA lead.

A good test case is the one which is effective at finding defects and also covers most of the scenarios on the system under test. Here is the step by step guide on how to develop a good test case :

- Test cases need to be simple and transparent
- Create test case with end user in mind
- Avoid test case repetition
- Do not assume functionality and features of your software application
- Ensure 100% coverage of software requirements
- Name the test case id such that they are identified easily while tracking defects
- Implement testing techniques
- The test case you create must return the Test Environment to the pre-test state
- The test case should generate the same results every time
- Your peers should be able to uncover defects in your test case design

Test Environment Setup

Setting up the test environment is vital part of the Software Testing Life Cycle. A testing environment is a setup of software and hardware for the testing teams to execute test cases. It supports test execution with hardware, software and network configured.



The test environment involves setting up of distinct areas like :

- **Setup of Test Server** – Every test may not be executed on a local machine. It may need establishing a test server, which can support applications.
- **Network** – We need to set up the network as per requirements.
- **Test PC Setup** – We need to set up different browsers for different testers.
- **Bug Reporting** – Bug reporting tools should be provided to testers.
- **Creating Test Data for the Test Environment** – Many companies use a separate test environment to test the software product. The common approach used is to copy production data to test.

Test Execution

The next phase in Software Testing Life Cycle is Test Execution. Test execution is the process of executing the code and comparing the expected and actual results. When test

execution begins, the test analysts start executing the test scripts based on test strategy allowed in the project.

Entry Criteria	Deliverable
• Test Plan or Test strategy document. • Test cases. • Test data.	• Test case execution report. • Defect report.

Activities
• Mark status of test cases like Passed, Failed, Blocked, Not Run etc. • Assign Bug Id for all failed and blocked test cases. • Do Retesting once the defects are fixed. • Track the defects to closure.

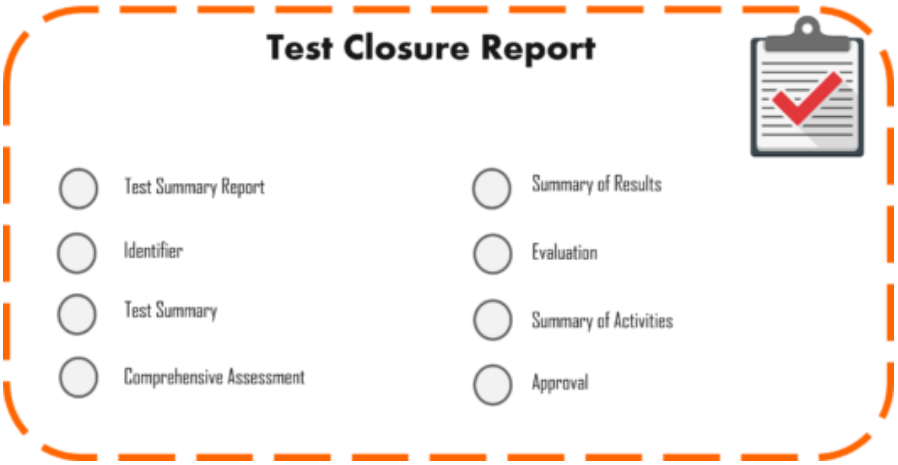
Test Cycle Closure

The final phase of the Software Testing Life Cycle is Test Cycle Closure. It involves calling out the testing team member meeting & evaluating cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software.

A test closure report by the test lead is published after accomplishing the exit criteria and finishing the testing phase. It follows a standard format such as :

- Test Summary Report
- Identifier
- Test Summary
- Variances
- Comprehensiveness Assessment
- Summary of Results

- Evaluation
- Summary of Activities
- Approval



Stages of Test Closure :

The process of test closure is implemented with the assistance of six important stages such as –

1. **Check planned Deliverable** – The planned deliverables that will be given to the stakeholder of the project are checked and analyzed by the team.
2. **Close Incident Reports** – The team checks that the planned deliverable are delivered and validates that all the incidents are resolved before the culmination of the process.
3. **Handover to Maintenance** – After resolving incidents and closing the incident report, the test-wares are then handed over to the maintenance team.
4. **Finalize & Archive Testware/Environment** – It involves finalizing and archiving of the testware and software like test scripts, test environment, test infrastructure, etc.
5. **Document System Acceptance** – It involves system verification and validation according to the strategy outlined.
6. **Analyze Best Practices** – It determines the various changes required for similar projects and their future release.

Let’s now move ahead with this article and understand the difference between SDLC and STLC.

What is SDLC and STLC in Software Testing?

SDLC	STLC
Stands for Software Development Life Cycle	Stands for Software testing Life Cycle
It refers to a sequence of various activities that are performed during the software development process	It refers to a sequence of various activities that are performed during the software testing process
Aims to complete the development of the software including testing and other phases successfully	Aims to evaluate the functionality of a software application to find any software bugs
In SDLC, the code for the software is built based on the design documents	In STLC, the test environment is created and various tests are carried out on the software

Now with this, we come to an end to this "Software Testing Life Cycle" blog. I hope you guys enjoyed this article and understood what is software testing and the different Types of Software testing.

What is Functional Testing?

FUNCTIONAL TESTING is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

Functional testing mainly involves black box testing and it is not concerned about the source code of the application. This testing checks User Interface, APIs, Database, Security, Client/Server communication and other functionality of the Application Under Test. The testing can be done either manually or using automation.

What do you test in Functional Testing?

The prime objective of Functional testing is checking the functionalities of the software system. It mainly concentrates on -

- **Mainline functions**: Testing the main functions of an application
- **Basic Usability**: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.
- **Accessibility**: Checks the accessibility of the system for the user
- **Error Conditions**: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

Software Testing Methodologies

Functional vs. Non-functional Testing

The goal of utilizing numerous testing methodologies in your development process is to make sure your software can successfully operate in multiple environments and across different platforms. These can typically be broken down between functional and non-functional testing.

FUNCTIONAL TESTING is a type of software testing that validates the software system against the functional requirements/specifications. The purpose of Functional tests is to test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.

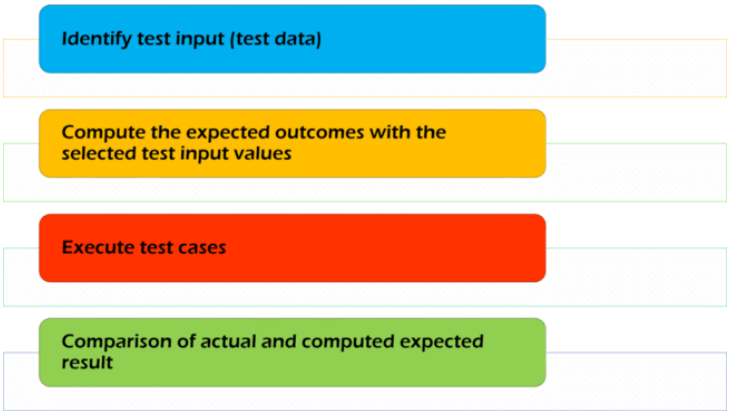
Functional testing involves testing the application against the business requirements. It incorporates all test types designed to guarantee each part of a piece of software behaves as expected by using uses cases provided by the design team or business analyst. These testing methods are usually conducted in order and include:

- Unit testing
- Integration testing
- System testing

- Acceptance testing

How to perform Functional Testing: Complete Process

In order to functionally test an application, the following steps must be observed.



- Understand the Software Engineering Requirements
- Identify test input (test data)
- Compute the expected outcomes with the selected test input values
- Execute test cases
- Comparison of actual and computed expected result

Non-functional testing methods incorporate all test types focused on the operational aspects of a piece of software. These include:

- Performance testing
- Security testing
- Usability testing
- Compatibility testing

The key to releasing high quality software that can be easily adopted by your end users is to build a robust testing framework that implements both functional and non-functional software testing methodologies.

Unit Testing

Unit testing is the first level of testing and is often performed by the developers themselves. It is the process of ensuring individual components of a piece of software at the code level are functional and work as they were designed to. Developers in a test-driven environment will typically write and run the tests prior to the software or feature being passed over to the test team. Unit testing can be conducted manually, but automating the process will speed up delivery cycles and expand test coverage. Unit testing will also make debugging easier because finding issues earlier means they take less time to fix than if they were discovered later in the testing process. TestLeft is a tool that allows advanced testers and developers to shift left with the fastest test automation tool embedded in any IDE.

Integration Testing

After each unit is thoroughly tested, it is integrated with other units to create modules or components that are designed to perform specific tasks or activities. These are then tested as group through integration testing to ensure whole segments of an application behave as expected (i.e, the interactions between units are seamless). These tests are often framed by user scenarios, such as logging into an application or opening files. Integrated tests can be conducted by either developers or independent testers and are usually comprised of a combination of automated functional and manual tests.

System Testing

System testing is a black box testing method used to evaluate the completed and integrated system, as a whole, to ensure it meets specified requirements. The functionality of the software is tested from end-to-end and is typically conducted by a separate testing team than the development team before the product is pushed into production.

Acceptance Testing

Acceptance testing is the last phase of functional testing and is used to assess whether or not the final piece of software is ready for delivery. It involves ensuring that the product is in compliance with all of the original business criteria and that it meets the end user's needs. This requires the product be tested both internally and externally, meaning you'll need to get it into the hands of your end users for beta testing along with those of your QA team. Beta testing is key to getting real feedback from potential customers and canaddress any final usability concerns.

Performance Testing

Performance testing is a non-functional testing technique used to determine how an application will behave under various conditions. The goal is to test its responsiveness and stability in real user situations. Performance testing can be broken down into four types:

- **Load testing** is the process of putting increasing amounts of simulated demand on your software, application, or website to verify whether or not it can handle what it's designed to handle.
  - **Stress testing** takes this a step further and is used to gauge how your software will respond at or beyond its peak load. The goal of stress testing is to overloadthe application on purpose until it breaks by applying both realistic and unrealistic load scenarios. With stress testing, you'll be able to find the failure point of your piece of software.
  - **Endurance testing,** also known as soak testing, is used to analyze the behavior of an application under a specific amount of simulated load over longer amounts of time. The goal is to understand how your system will behave under sustained use, making it a longer process than load or stress testing (which are designed to end after a few hours). A critical piece of endurance testing is that it helps uncover memory leaks.
  - **Spike testing** is a type of load test used to determine how your software will respond to substantially larger bursts of concurrent user or system activity over varying amounts of time. Ideally, this will help you understand what will happen when the load is suddenly and drastically increased.

Security Testing

With the rise of cloud-based testing platforms and cyber attacks, there is a growing concern and need for the security of data being used and stored in software. Security testing is a non-functional software testing technique used to determine if the information and data in a system is protected. The goal is to purposefully find loopholes and security risks in the system that could result in unauthorized access to or the loss of information by probing the application for weaknesses. There are multiple types of this testing method, each of which aimed at verifying six basic principles of security:

1. Integrity
2. Confidentiality
3. Authentication
4. Authorization
5. Availability
6. Non-repudiation

Usability Testing

Usability testing is a testing method that measures an application's ease-of-use from the end-user perspective and is often performed during the system or acceptance testing stages. The goal is to determine whether or not the visible design and aesthetics of an application meet the intended workflow for various processes, such as logging into an application. Usability testing is a great way for teams to review separate functions, or the system as a whole, is intuitive to use.

Compatibility Testing

Compatibility testing is used to gauge how an application or piece of software will work in different environments. It is used to check that your product is compatible with multiple operating systems, platforms, browsers, or resolution configurations. The goal is toensure that your software's functionality is consistently supported across any environment you expect your end users to be using.

Functional Vs Non-Functional Testing:

Functional Testing	**Non-Functional Testing**
Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements.	Non-Functional testing checks the Performance, reliability, scalability and other non-functional aspects of the software system.
Functional testing is executed first	Non-functional testing should be performed after functional testing
Manual Testing or automation tools can be used for functional testing	Using tools will be effective for this testing
Business requirements are the inputs to functional testing	Performance parameters like speed, scalability are inputs to non-functional testing.
Functional testing describes what the product does	Nonfunctional testing describes how good the product works
Easy to do Manual Testing	Tough to do Manual Testing

Examples of Functional testing are	Examples of Non-functional testing are
• Unit Testing • Smoke Testing • Sanity Testing • Integration Testing • White box testing • Black Box testing • User Acceptance testing • Regression Testing	• Performance Testing • Load Testing • Volume Testing • Stress Testing • Security Testing • Installation Testing • Penetration Testing • Compatibility Testing • Migration Testing

Defect

A Software **DEFECT / BUG / FAULT** is a condition in a software product which does not meet a software requirement (as stated in the requirement specifications) or end-user expectation (which may not be specified but is reasonable). In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/ unexpected results.

- **defect:** An imperfection or deficiency in a work product where it does not meet its requirements or specifications.

Related Terms

- A program that contains a large number of bugs is said to be ***buggy***.
- Reports detailing defects / bugs in software are known as ***defect reports / bug reports***.
- Applications for tracking defects bugs are known as ***defect tracking tools / bug tracking tools***.

- The process of finding the cause of bugs is known as ***debugging***.
- The process of intentionally injecting bugs in a software program, to estimate test coverage by monitoring the detection of those bugs, is known as ***bebugging***.

Software Testing proves that defects exist but NOT that defects do not exist.

Classification

Software Defects/ Bugs are normally classified as per:

- Severity / Impact
- Probability / Visibility
- Priority / Urgency
- Related Dimension of Quality
- Related Module / Component
- Phase Detected
- Phase Injected

Related Module /Component

Related Module / Component indicates the module or component of the software where the defect was detected. This provides information on which module / component is buggy or risky.

- Module/Component A
- Module/Component B
- Module/Component C
- …

Phase Detected

Phase Detected indicates the phase in the software development lifecycle where the defect was identified.

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

Phase Injected

Phase Injected indicates the phase in the software development lifecycle where the bug was introduced. Phase Injected is always earlier in the software development lifecycle than the Phase Detected. Phase Injected can be known only after a proper root-cause analysis of the bug.

- Requirements Development
- High Level Design
- Detailed Design
- Coding
- Build/Deployment

Note that the categorizations above are just guidelines and it is up to the project/ organization to decide on what kind of categorization to use. In most cases, the categorization depends on the defect tracking tool that is being used. It is essential that project members agree beforehand on the categorization (and the meaning of each categorization) so as to avoid arguments, conflicts, and unhealthy bickering later.

Metrics

Some metrics related to Defects are:

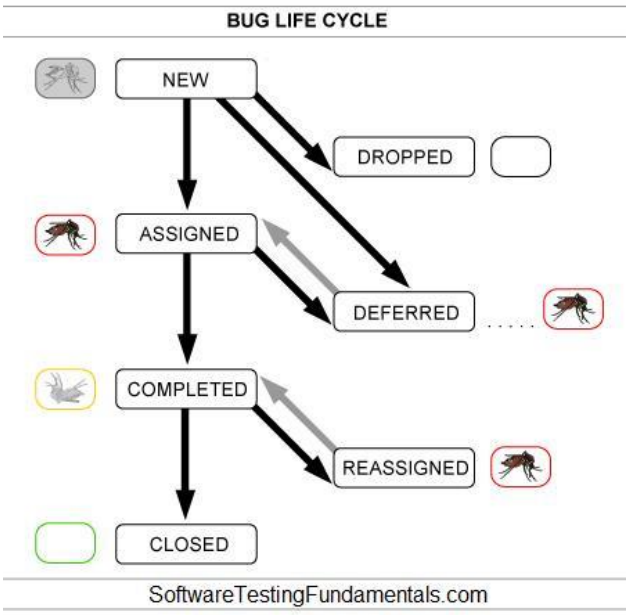
- Defect Age
- Defect Density
- Defect Detection Efficiency

Defect vs Bug

Strictly speaking, a BUG is a deficiency in just the software but a DEFECT could be a deficiency in the software as well as any work product (Requirement Specification, for example). You don't say 'There's a bug in the Test Case'; you say 'There's a defect in the Test Case.'We prefer the term 'Defect' over the term 'Bug' because 'Defect' is more comprehensive.

Defect Life Cycle

DEFECT LIFE CYCLE, also known as Bug Life Cycle, is the journey of a defect from its identification to its closure. The Life Cycle varies from organization to organization and is governed by the software testing process the organization or project follows and/or the Defect tracking tool being used. Nevertheless, the life cycle in general resembles the following:



Status

Status	Alternative Status
NEW	
ASSIGNED	**OPEN**
DEFERRED	
DROPPED	**REJECTED**
COMPLETED	**FIXED, RESOLVED, TEST**
REASSIGNED	**REOPENED**
CLOSED	**VERIFIED**

- *NEW*: Tester finds a defect and posts it with the status NEW. This defect is yet to be studied/approved. The fate of a NEW defect is one of ASSIGNED, DROPPED or DEFERRED.
- *ASSIGNED / OPEN*: Test / Development / Project lead studies the NEW defect and if it is found to be valid it is assigned to a member of the Development Team. The assigned Developer's responsibility is now to fix the defect and have it COMPLETED. Sometimes, ASSIGNED and OPEN can be different statuses. In that case, a defect can be open yet unassigned.
- *DEFERRED*: If a valid NEW or ASSIGNED defect is decided to be fixed in upcoming releases instead of the current release it is DEFERRED. This defect is ASSIGNED when the time comes.

- *DROPPED / REJECTED*: Test / Development/ Project lead studies the NEW defect and if it is found to be invalid, it is DROPPED / REJECTED. Note that the specific reason for this action needs to be given.
- *COMPLETED / FIXED / RESOLVED / TEST*: Developer 'fixes' the defect that is ASSIGNED to him or her. Now, the 'fixed' defect needs to be verified by the Test Team and the Development Team 'assigns' the defect back to the Test Team. A COMPLETED defect is either CLOSED, if fine, or REASSIGNED, if still not fine.
- If a Developer cannot fix a defect, some organizations may offer the following statuses:
- *Won't Fix / Can't Fix*: The Developer will not or cannot fix the defect due to some reason.
- *Can't Reproduce*: The Developer is unable to reproduce the defect.
- *Need More Information*: The Developer needs more information on the defect from the Tester.
- *REASSIGNED / REOPENED*: If the Tester finds that the 'fixed' defect is in fact not fixed or only partially fixed, it is reassigned to the Developer who 'fixed' it. A REASSIGNED defect needs to be COMPLETED again.
- *CLOSED / VERIFIED*: If the Tester / Test Lead finds that the defect is indeed fixed and is no more of any concern, it is CLOSED / VERIFIED. This is the happy ending.

Guidelines

- Make sure the entire team understands what each defect status exactly means. Also, make sure the defect life cycle is documented.
- Ensure that each individual clearly understands his/her responsibility as regards each defect.
- Ensure that enough detail is entered in each status change. For example, do not simply DROP a defect but provide a reason for doing so.

- If a defect tracking tool is being used, avoid entertaining any 'defect related requests' without an appropriate change in the status of the defect in the tool. Do not let anybody take shortcuts. Or else, you will never be able to get up-to-date and reliable defect metrics for analysis.

Verification and Validation

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose. **Barry Boehm** described verification and validation as the following:

Verification: *Are we building the product right?*

Validation: *Are we building the right product?*

Verification:

Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.

Verification is **Static Testing**.

Activities involved in verification:

1. Inspections
2. Reviews
3. Walkthroughs
4. Desk-checking

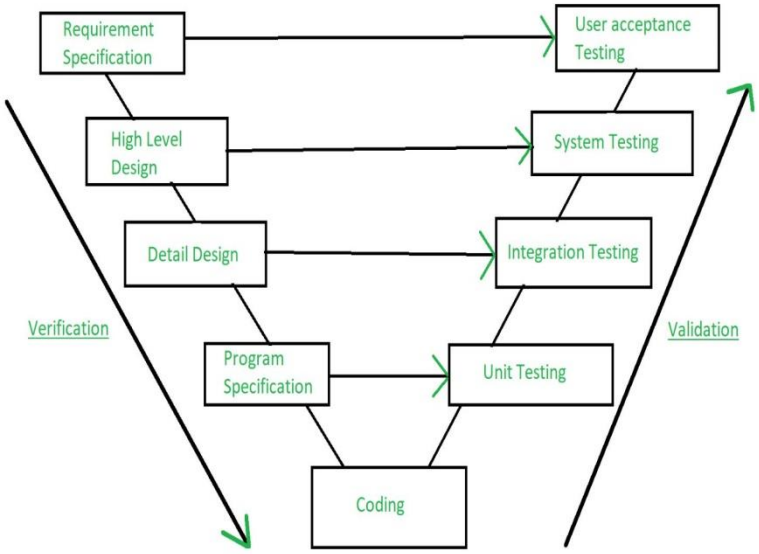
Validation:

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expectedproduct.

Validation is the **Dynamic Testing**.

Activities involved in validation:

1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing



Note: Verification is followed by Validation.

Software Testing Functional and Structural

Functional testing is sometimes called black-box testing because no knowledge of the internal logic of the system is used to develop test cases. For example, if a certain function key should produce a specific result when pressed, a functional test validates this expectation by pressing the function key and observing the result. When conducting functional tests, you'll use validation techniquesalmost exclusively.

Conversely, structural testing is sometimes called white-box testing because knowledge of the internal logic of the system is used to develop hypothetical test cases. Structural tests use verification predominantly. If a software development team creates a block of code that will

allow a system to process information in a certain way, a test team would verify this structurally by reading the code, and given the system's structure, see if the code would work reasonably. If they felt it could, they would plug the code into the system and run an application to structurally validate the code.

Each method has its pros and cons:

Functional Testing Advantages

1• Simulates actual system usage.
2• Makes no system structure assumptions

Functional Testing Disadvantages

1• Potential of missing logical errors in software
2• Possibility of redundant testing

Structural Testing Advantages

1• You can test the software's structure logic
2• You test code that you wouldn't use if you performed only functional testing

Structural Testing Disadvantages

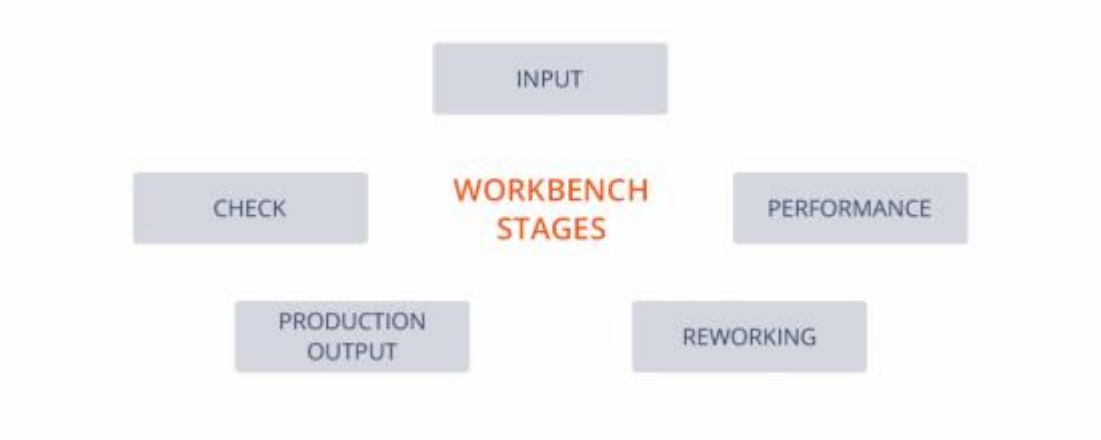
1• Does not ensure that you've met user requirements
2• Its tests may not mimic real-world situations

A functional test case might be taken from the documentation description of how to perform a certain function, such as accepting bar code input. A structural test case might be taken from a technical documentation manual. To effectively test systems, you need to use both methods.

Test Phase	Performed by:	Verification	Validation
Feasibility Review	Developers, Users	X	
Requirements Review	Developers, Users	X	
Unit Testing	Developers		X
Integrated Testing	Developers		X
System Testing	Developers with User Assistance		X

Workbench

This is a method which aims to examine and verify the structure of testing performance by detailed documenting. Workbench process has its common stages and steps which serve for different test assignments. The common stages of each workbench include:



Input. It is the initial workbench stage. Each certain assignment should contain its initial and outcome (input and output) requirements to know the available parameters and expected results. Each workbench has its specific inputs depending on the type of product under testing.

Performance. The priority aim of the entire testing is in the transformation of the initial parameters to outcome requirements and reach the prescribed results.

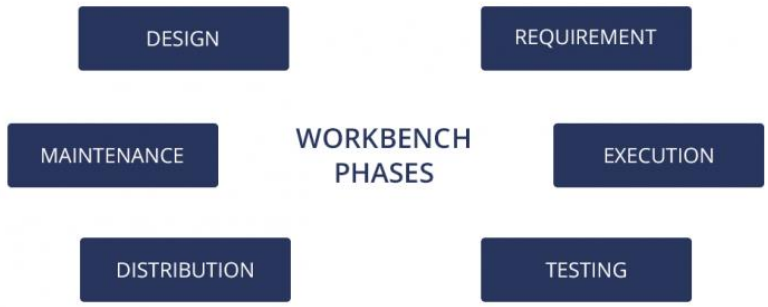
Check. It is an examination of output parameters after the performance phase to verify its accordance with the expected ones.

Production output. It is the final stage of a workbench in case the check confirmed the properly conducted performance.

Reworking. If the outcome parameters are not in compliance with the desired result, it is necessary to return to the performance phase and conduct it from the beginning.

Workbench Phases

Let's look at the initial and outcome data from different angles, considering various phases:



Requirement phase

- the initial data should be collected from the customer to perform a test task;
- the customer's requirements are included in the document to check its accordance with clients' needs;
- the outcome data are received and codified in one document.

Design phase

- the initial data is in the requirement document;
- testers prepare the technical document and check if the design document is technically proper;
- testers check if the information about the requirements is transferred to the requirement document completely.

Execution phase

- it is the performance of the entire testing process;
- the initial data is contained in the technical document, and the performance means adjusting of code according to the documented technical requirements;

- the outcome data is the source code.

Testing phase

The initial parameters are contained in the source code, and the outcomes are formed after the test performance.

Distribution phase

The aim of this phase is to prepare the product which is ready for use. Initial data, in this case, is a code version given by customer with initial requirements and code after testing.

Maintenance phase

- input appears as the outcome of distribution;
- the outcome data forms a new release;
- each change in product requirements is subjected to regression testing to fulfill the customers' requests.

The workbench concept serves to build and monitor the proper structure of testers' work. It helps to divide assignments in each phase of testing and reach the customers' expectations relying on initial data and transforming the product parameters into desirable ones.

The eight considerations listed below provide the framework for developingtesting tactics.

Each is described in the following sections.

- Acquire and study the test strategy
- Determine the type of development project
- Determine the type of software system
- Determine the project scope
- Identify the tactical risks
- Determine when testing should occur
- Build the tactical test plan

•Build the unit test plansAcquire and Study the Test Strategy

A team familiar with the business risks associated with the software normallydevelops the test strategy, and the test team develops the tactics. Thus, the testteam needs to acquire and study the test strategy, focusing on the followingquestions:

•What is the relationship of importance among the test factors?

•Which of the high-level risks are the most significant?

•Who has the best understanding of the impact of the identified businessrisks?

•What damage can be done to the business if the software fails to performcorrectly?

•What damage can be done to the business if the software is notcompleted on time?

Determine the Type of Development Project

The type of project refers to the environment in which the software willbedeveloped, and the methodology used. Changes to the environment also changethe testing risk. For example, the risks associated with a traditional developmenteffort are different from the risks associated with off-the shelf purchasedsoftware.

Testing checklist

During SDLC (Software Development Life Cycle) while software is in the testing phase, it is advised to make a list of all the required documents and tasks to avoid last minute hassle. This way tester will not miss any important step and will keep a check on quality too. If the tester doesn't make any checklist or forgets to include any task in it then it is possible that he may miss some of the important defects.

Testing Checklist is divided into number of categories which are listed as follows:

1) Resource Assignment and Training

- To make sure that testing project has sufficient budget allocated at project level.
- We have sufficient staffing or human resources allocated for testing project.
- Analyze skills and competencies of test team to make sure whether they are competent enough or required more grooming to meet required skill set.
- All required testing tools are installed at workstation with appropriate software licence.
- All resources are well trained on required testing tools and project business.
- Required responsibilities are assigned to team member and respective leads.
- All required sign off are procured from senior management for staffing and training.

2) Software Testing Documentation

- Make sure that all the functional documents and design documents are completed before testing team can start writing test cases.
- Test plan is created covering all the required test cases.
- Test cases are created covering all the required business use cases.
- Review of test cases and test plan following maker and checker policy.
- Setting up of Bug reporting portal to log the defects.
- Creation of tractability matrix with the functional team to make sure functions are mapped to test cases.
- Make sure, project weekly status report format is well defined.
- Sign off or approval from QA manager to execute the test cases.

3) Software Testing Checklist

- Regression suite is executed successfully when testing with new test phase or new project release.
- Make sure each tester is filling the time sheet and logging defect in defect portal on daily basis.
- Keeping a check on total test cases executed on daily basis and hence project work progress.
- Test weekly status reports is circulated on weekly basis with correct format and to required recipients.

- Open bugs are addressed timely by development team, requirement gathering team and senior management.
- Make sure, there are no roadblocks in testing area related to technologies, management and client behavior.
- Make sure before declaring the testing status as complete or providing testing sign off, all major or minor open bugs or defects are either closed or deferred for future release.
- Make sure all system compatibility checks are done, e.g. an application working on IE explorer should also work on chrome, Mozilla, etc.

4) Compliance's

- Review of test plan to make sure project complies with the required design methods.
- Evaluation of project goal statement with the business use cases.
- Project should comply with all required legal compliance's.
- Identify the priorities items in the project which are necessary for organization compliance's and execute those items.
- Examine project plan for strategic compliance with objective of business organization.
- Verify that project deliverable are in compliance with the client requirements.
- Evaluate project capabilities meets the desired outcome to accomplish predefined project goal.
- All necessary project compliance sign offs are procured from senior management.

5) Measurability and Monitoring

- Evaluation of project activities and processes that are well measurable to set the desired level of performance.
- Verification of System process measures for reliability and accuracy.
- Requesting the client to assign accessibility of project system to project team for review.
- Scheduling check point call, test phase completion call and daily scrum call to monitor the progress of test project.
- Make sure test project deliverable has predefined acceptance criteria which is approved, this will help to measure project deliverable.

- Make sure, proper escalation contact details are well communicated to client and project team members.

6) Project Flexibility

- Make sure that project is flexible and has ability to make desired amendments timely.
- Evaluate project has risk mitigation plan after analyzing all the possible project risk factors.
- Make sure project control system is reliable and effective.
- Make sure that project has a contingency plan to address exception and unforeseen events.
- Be confident that project goal completely address problems defined by the business use cases.
- Creation of self-regulatory feedback loop for the project to make sure every problem is reported related to test project work.

These are some of the main terms should be included in the **Testing Checklist**, however, every organization has different software and application *Testing Checklist* may vary. It is always a good practice to make a checklist so that testing can be done in a proper way and no important point should be missed.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

School of Computing

Department of Computer Science and Engineering

and

Department of Information Technology

SCS1608 - Software Quality Assurance and Testing

UNIT - IV

Black-box testing

Black box diagram

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it.[2] Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing.

Specification-based testing aims to test the functionality of software according to the applicable requirements.[3] This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case. Test cases are built around specifications and requirements, i.e., what the application is supposed to do. It uses external descriptions of the software, including specifications, requirements, and designs to derive test cases. These tests can be functional or non-functional, though usually functional.

Specification-based testing may be necessary to assure correct functionality, but it is insufficient to guard against complex or high-risk situations.[4]

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality. On the other hand, black-box testing has been said to be "like a walk in a dark labyrinth without a flashlight."[5] Because they do not examine the source code, there are situations when a tester writes many test cases to check something that could have been tested by only one test case, or leaves some parts of the program untested.

This method of test can be applied to all levels of software testing: unit, integration, system and acceptance. It typically comprises most if not all testing at higher levels, but can also dominate unit testing as well.

Boundary Value Analysis

Boundary value analysis, BVA, tests the behavior of a program at the boundaries. When checking a range of values, after selecting the set of data that lie in the valid partitions, next is to check how the program behaves at the boundary values of the valid partitions. Boundary value analysis is most common when checking a range of numbers.

For each range, there are two boundaries, the lower boundary (start of the range) and the upper boundary (end of the range) and the boundaries are the beginning and end of each valid partition. We should design test cases which exercises the program functionality at the boundaries, and with values just inside and just outside the boundaries.

Here, the assumption is that if the program works correctly for these extreme cases, then it will work correctly for all values in between the valid partition. Testing has shown that defects that arise when checking a range of values the most defects are near or at the boundaries.

Boundary Value Analysis Example

A program which accepts an integer in the range **-100** to **+100**, there would be three sets of valid equivalent partitions: these are -10 to -1, the negative range 0, Zero and 1 to 10, the positive range.

For each range, there are minimum and maximum values at each boundary. For the negative range the lower boundary is -10 and the upper boundary is -1. At each boundary, three conditions should be checked.

-101, **-100**, -99 … … … … … … -2, **-1**,

0, -1, **0**, +1

0, **1**, 2 … … … … … … 99, **100**, 101

You might have noticed that by selecting values at the boundaries for each partition there are some values that overlap. That is they would appear in our test conditions when we check the

boundaries. We should of course make redundant those values that overlap to eliminate unnecessary test cases.

Another worthy note to consider is that because the range goes from -100 to +100, then effectively, the "boundary values" -2, -1 and -99 are considered as equivalent. That is, the behavior of the system is the same (or should be the same) when testing with values -99, -2 and -1.

However, because -99 is close to the upper limit boundary, and is most likely to reveal defects, we keep the value -99 in our test case and scrap conditions for -2 and -1. The same applies to the positive range boundaries, that is, +1, +2 and +99 are expected to give the same behavior. However, because 99 is close to the upper range boundary, we keep 99 in our list of data to test and get rid of values 1 and 2.

Therefore, our list of data to check at the boundaries become

-101, **-100**, -99 … … … … … … 0 … … … … … … 99, **100**, 101

If our range of data was from 0 to +10 then the boundary value analysis would give us the following values to test:

-1, **0**, 1 … … … … … …9, **10**, 11

Here, 1 and 9 are both part of the test condition because each are at the boundary of the whole range and can reveal defects.

It is important to note that that boundary values analysis can be applied to float numbers as well as integers. The only difference is that when analyzing boundaries for float numbers, we should test to the closest decimal point.

Example, if we have a range from 5.5 to 9.9, then the set of data at boundaries becomes

5.4, **5.5**, 5.6 … … … … … … 9.8, **9.9**, 10.0

Like Equivalence Partitioning Test Technique, Boundary Value Analysis is a common black box testing technique and is one which must be applied when checking a range of values. Together

with equivalence partitioning and negative testing, it can be a very powerful black box testing technique to find defects when testing a range of values.

INTEGRATION TESTING

Integration Test Case

Integration Test Case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links**rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email should appear the Deleted/Trash folder

Approaches/Methodologies/Strategies of Integration Testing:

Software Engineering defines variety of strategies to execute Integration testing, viz.

- Big Bang Approach :
- Incremental Approach: which is further divided into the following

- Top Down Approach
- Bottom Up Approach
- Sandwich Approach - Combination of Top Down and Bottom Up

Below are the different strategies, the way they are executed and their limitations as well advantages.

Big Bang Approach:

Here all component are integrated together at **once** and then tested.

Advantages:

- Convenient for small systems.

Disadvantages:

- Fault Localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
- Since the Integration testing can commence only after "all" the modules are designed, the testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Approach

In this approach, testing is done by joining two or more modules that are ***logically related***. Then the other related modules are added and tested for the proper functioning. The process continues until all of the modules are joined and tested successfully.

Incremental Approach, in turn, is carried out by two different Methods:

- Bottom Up

- Top Down

What is Stub and Driver?

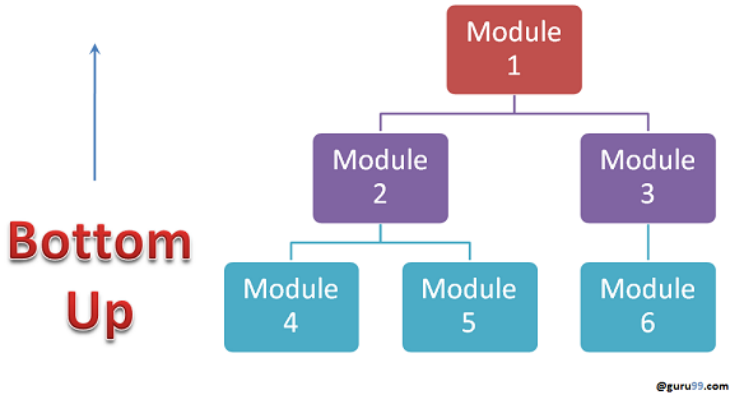
Incremental Approach is carried out by using dummy programs called **Stubs and Drivers**. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

Stub: Is called by the Module under Test.

Bottom-up Integration

In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

Diagrammatic Representation:



Advantages:

- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.

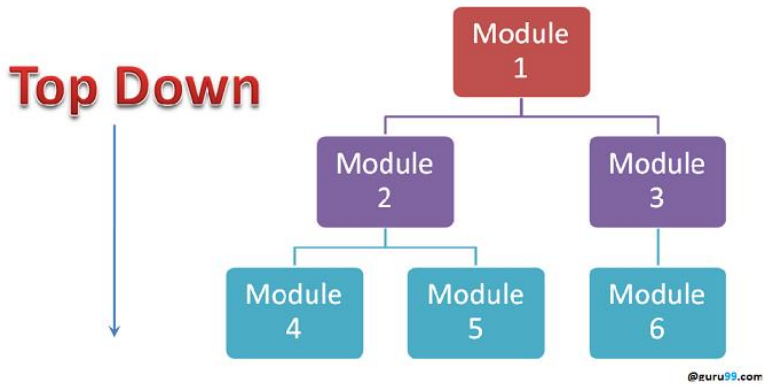
- An early prototype is not possible

Top-down Integration:

In Top to down approach, testing takes place from top to down following the control flow of the software system.

Takes help of stubs for testing.

Diagrammatic Representation:



Advantages:

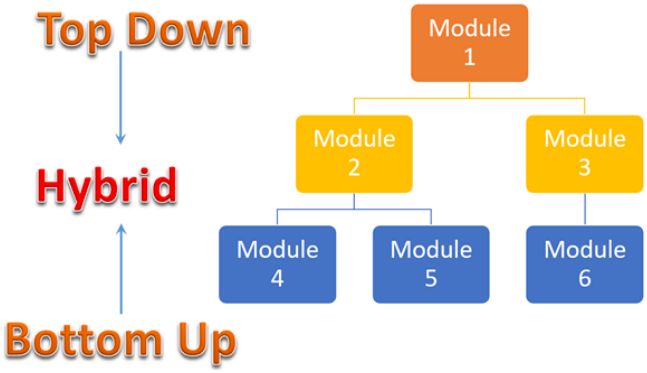
- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- Needs many Stubs.
- Modules at a lower level are tested inadequately.

Hybrid/ Sandwich Integration

In the sandwich/hybrid strategy is a combination of Top Down and Bottom up approaches. Here, top modules are tested with lower modules at the same time lower modules are integrated with top modules and tested. This strategy makes use of stubs as well as drivers.



Branch Coverage

In the branch coverage, every outcome from a code module is tested. For example, if the outcomes are binary, you need to test both True and False outcomes.

It helps you to ensure that every possible branch from each decision condition is executed at least a single time.

By using Branch coverage method, you can also measure the fraction of independent code segments. It also helps you to find out which is sections of code don't have any branches.

The formula to calculate Branch Coverage:

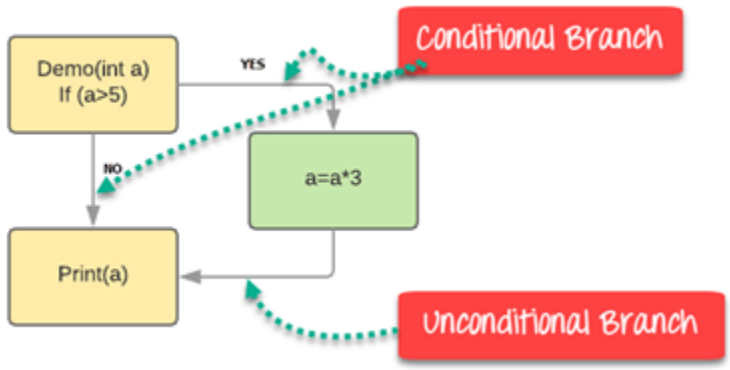
$$Branch\ Coverage = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches}$$

Example of Branch Coverage

To learn branch coverage, let's consider the same example used earlier

Consider the following code

```
Demo(int a) {
    If (a> 5)
            a=a*3
    Print (a)
   }
```



Branch Coverage will consider unconditional branch as well

Test Case	Value of A	Output	Decision Coverage	Branch Coverage
1	2	2	50%	**33%**
2	6	18	50%	**67%**

Advantages of Branch coverage:

Branch coverage Testing offers the following advantages:

- Allows you to validate-all the branches in the code
- Helps you to ensure that no branched lead to any abnormality of the program's operation

- Branch coverage method removes issues which happen because of statement coverage testing
- Allows you to find those areas which are not tested by other testing methods
- It allows you to find a quantitative measure of code coverage
- Branch coverage ignores branches inside the Boolean expressions

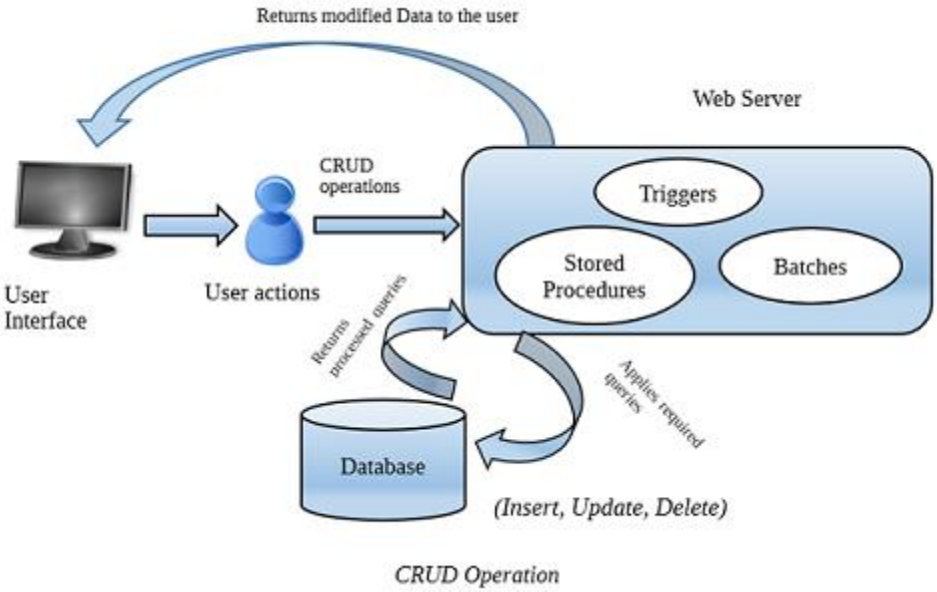
CRUD Testing

CRUD testing is a black box testing. CRUD is an acronym for Create, Read, Update, Delete. CRUD testing is another term for database testing. Database forms an inevitable part of a software. Database forms the backbone of any application- web or desktop, data is stored somewhere. A user feeds in the information and the data gets stored in some form. An organisation may use any of the available databases available in the market - MS Access, Oracle, MySQL, DB2 etc. It depends on the needs and requirements of an organisation as to which kind of database has to be integrated with their application. Every database has its own set of features and associated cost with it.

CRUD operations from the perspective of an end-user can thought of like this -

- Create - user 'Saving' any new transaction.
- Read/Retrieve - User 'Searching' or 'Viewing' any transaction.
- Update - when a user wants to 'Edit' or 'Modify' an existing data.
- Delete - when user wants to 'Remove' any data from the system.

Let's take up a very common scenario of any of the social networking application. When a user is new, he registers on the application and the data gets stored in the application server. If in case someone wants to update their information or add new information, they have the required option. Examples can be skype, facebook etc.



CRUD Operation

Why Database Testing :

- **To ensure Data Mapping -**The system must be able to track the records correctly. That means, respective values are updated in the tables. For instance person A wants to change his name and phone number that should reflect in appropriate record fields.

- **Maintain ACID Properties -** ACID properties lie at the core of a database structure. Atomicity, consistency, isolation, durability - these four components are essential and forms a complete package for a robust database. *Atomicity* means that changes must reflect all across uniformly or none should be affected. *Consistency* implies that a data must only be updated when it follows some predefined rules. *Isolation* determines how users are able to view the transactions, that is, isolation rules define how a change made to a certain thing is reflected all across. *Durability* states the transactions once committed will be saved permanently . For instance, if a booking has been done for a flight, the booking for a seat will remain even if there is some mishap in the system.

- **Ensure Data Integrity -**Data integrity means when a data is modified, the latest state of data must reflect in every system. A system therefore must show the updated and most recent values to all the users accessing it from anywhere.

- **Ensure Accuracy of implemented business Rules -**In the current scenario database follow the concept of RDBMs which is relational database management system. Referential integrity refers to the relation that exists among the tables. Primary key contraints, foreign keys, triggers, stored procedures etc form a set of business rules. Therefore developers integrate these business rules into their code when implementing database business logic. These business rules are essential in maintaining the ACID properties.

Grey Box Testing?

Grey Box testing is testing technique performed with limited information about the internal functionality of the system. Grey Box testers have access to the detailed design documents along with information about requirements.

Grey Box tests are generated based on the state-based models, UML Diagrams or architecture diagrams of the target system.

Gray-box testing Techniques:

- Regression testing
- Pattern Testing
- Orthogonal array testing
- Matrix testing

Benefits:

- Grey-box testing provides combined benefits of both white-box and black-box testing
- It is based on functional specification, UML Diagrams, Database Diagrams or architectural view
- Grey-box tester handles can design complex test scenario more intelligently

- The added advantage of grey-box testing is that it maintains the boundary between independent testers and developers

Drawbacks:

- In grey-box testing, complete white box testing cannot be done due to inaccessible source code/binaries.
- It is difficult to associate defects when we perform Grey-box testing for a distributed system.

Best Suited Applications:

Grey-box testing is a perfect fit for Web-based applications.

Grey-box testing is also a best approach for functional or domain testing.

JAD

JAD (Joint Application Development) is a software development approach which engages the client and/or the end users for designing and developing the system. This model was designed and put forward by Dr. Chuck Morris and Dr. Tony Crawford of IBM, who propose this model in the late 1970s. As compared to other primitive SDLC model, Joint Application Development model leads to faster progression of the system development which has better client approval.

This model furthermore, is vast when it comes to agile delivery wherein the software products need to be developed as well as shipped in short iterations depending on agreements among the industrial as well as industry stakeholders which are termed as Minimum Viable Product (MVP).

Phases of jad

Since you have become familiar with the JAD concept, it is time to know about its phases and how the model's design and development approach works:

1. Define Specific Objectives: The facilitator, in partnership with stakeholders, set all the objectives as well as a list of items which is then distributed to other developers and participants to understand and review. This objective contains elements like the scope of this projected system, its potential outcome, technical specification required, etc.

2. Session Preparation: The facilitator is solely responsible for this preparation where all relevant data is collected and sent to other members before time. For better insight, research carried out to know about the system requirement better and gather all the necessary information for development.

3. Session Conduct: Here the facilitator is accountable to identify those issues which have to be working out for making the system error-free. Here the facilitator will serve as a participant but will not have a say regarding any information.

4. Documentation: After the product is developed, the records and published documents are put forward into the meeting so that the stakeholders and consumers can approve it through the meeting.

Benefits of jad

- Improved Delivery Time: The time required for developing a product using JAD model is lesser and efficient than that of other traditional models.
- Cost Reduction: Efficiently analyzing the requirements and facts with business executives and stakeholders will make less effort to develop the system and hence less cost will be required for the entire development process.
- Better Understanding: Since the entire requirement is analyzed by business executives, followed by a cautious choice of developers and team member who can professionally interact with each other better usually helps in understanding the product development better.
- Improved Quality: Since all the key decision makers and stakeholders of the project are involved in the development of the project so there is the least chance of error and hence the product quality becomes better and more accurate.

Pareto Analysis?

- A statistical technique for making decisions which is used for selecting a limited number of tasks which produce significant overall effect. Pareto Analysis uses the ‘Pareto Principle’ – an

idea by which 80% of doing the entire job is generated by doing 20% of the work.

- When many possible courses of actions are completing the attention, the technique 'Pareto Analysis' is useful. In essence, the delivered benefit by each action is estimated by problem-solver, and selects the number of most effective actions which delivers the total benefit.

- Pareto Analysis is a mechanism to find changes that will give the most beneficial results. It states that only few factors are responsible for producing most problems. It is used so that the team can concentrate on those changes.

Prototype Testing?

Prototype Testing is conducted with the intent of finding defects before the website goes live. Online Prototype Testing allows seamlessly to collect quantitative, qualitative, and behavioural data while evaluating the user experience.

Characteristics of Prototype Testing:

- To evaluate new designs prior to the actual go live to ensure that the designs are clear, easy to use and meet users requirements.
- Is best when iterative testing is built into the development process, so that changes can be easily made often to ensure that major issues do not arise well before going live.
- Provides confirmation about the new design direction, branding and messaging are going in the right direction.

Random Testing?

Random Testing, also known as monkey testing, is a form of functional black box testing that is performed when there is not enough time to write and execute the tests.

Random Testing Characteristics:

- Random testing is performed where the defects are NOT identified in regular intervals.
- Random input is used to test the system's reliability and performance.

- Saves time and effort than actual test efforts.
- Other Testing methods Cannot be used to.

Random Testing Steps:

- Random Inputs are identified to be evaluated against the system.
- Test Inputs are selected independently from test domain.
- Tests are Executed using those random inputs.
- Record the results and compare against the expected outcomes.
- Reproduce/Replicate the issue and raise defects, fix and retest.

There are 2: Smart and Dump

Smart Monkeys – A smart monkey is identified by the below characteristics:-

- Have a brief idea about the application
- They know where the pages of application will redirect to.
- They know that the inputs they are providing are valid or invalid.
- They work or focus to break the application.
- In case they find an error, they are smart enough to file a bug.
- They are aware of the menus and the buttons.
- Good to do stress and load testing.

Dumb Monkey – A dumb monkey is identified by the below characteristics:

- They have no idea about the application.
- They don't know that the inputs they are providing are valid or invalid.
- They test the application randomly and are not aware of any starting point of the application or the end to end flow.
- Though they are not aware of application, but they too can identify bugs like environmental failure or hardware failure.
- They don't have much idea about the UI and functionality

Advantages of Monkey Testing:

- Can identify some out of the box errors.
- Easy to set up and execute

- Can be done by “not so skilled” resources.
- A good technique to test the reliability of the software
- Can identify bugs which may have higher impact.
- Not costly

Disadvantages of Monkey test:

- This can go on for days till a bug is not discovered.
- Number of bugs are less
- Reproducing the bugs (if occurs) becomes a challenge.
- Apart of some bugs, there can be some “Not Expected” output of a test scenario, analysis of which becomes a difficult and time consuming.

RISK BASED TESTING

Risk Based Testing (RBT) is a testing process with unique features. It is basically for those project and application that is based on risk. Using risk, Risk based testing prioritize and emphasize the suitable tests at the time of test execution. In other word, Risk is the chance of event of an unwanted outcome. This unwanted outcome is also related with an impact. Some time it is difficult to test all functionality of the application or it is might not possible. Use Risk based testing in that case; it tests the functionality which has the highest impact and probability of failure.

It’s better to start risk based testing with product risk analysis. There are numerous methods used for this are,

- Clear understanding of software requirements specification, design documents and other documents.
- Brainstorming with the project stakeholders.

  Risk-based testing is the process to understand testing efforts in a way that reduces the remaining level of product risk when the system is developed,
- Risk-based testing applied to the project at very initial level, identifies risks of the project that expose the quality of the project, this knowledge guides to testing planning, specification, preparation and execution.

- Risk-based testing includes both mitigation (testing to give chances to decrease the likelihood of faults, specially high-impact faults) and contingency (testing to know work-around to create the defects that do get past us less painful).
- Risk-based testing also includes measurement process that recognizes how well we are working at finding and removing faults in key areas.
- Risk-based testing also uses risk analysis to recognize proactive chances to take out or avoid defects through non-testing activities and to help us select which test activities to perform.

Major processes to execute the Risk-based testing are described below:

- Process 1 – Describe all requirements in terms of Risk involved in the project
- Process 2 – In terms of risk assessment, prioritize the requirements
- Process 3 – Plan and define tests according to requirement prioritization
- Process 4 – Execute test according to prioritization and acceptance criteria.

Process1– Projects associates various risks that are identified by the stakeholders of the project. The stake holders are basically a mixture of business and technical team. Stake holders involve various people from various departments for example, the client, customers, business experts, technical experts, project manager, project Leader, users, developers and infrastructure representative.

Process 2 – Once all the possible risks and their impacts are analyzed, the project manager has to get the requirements prioritized. Priority of the requirements should be agreed upon and should be updated in functional requirement document; the same should also be conveyed to the development and the test team.

Process 3 – After getting the Requirement with the priority tagged, we can start the test activities with keeping the priority of the requirements in mind.

Process 4 – If any of the identified risk realizes by the time of "Test execution Schedule", then there is quite good chances of schedule slippage from development side. In this case, the final deadline given to the customer can't be changed. In this situation, again, Test manager has to apply the Pareto principle and finalizes the scope of testing in the reduced timeline that will ensure least risk and highest quality.

Advantages

- Improved quality – All of the critical functions of the application are tested. Real time clear understanding of project risk.
- Give more focus on risks of the business project instead of the functionality of the information system.
- Provides a negotiating instrument to client and test manager similar when existing means are limited.
- Associate the product risk to the requirement identifies gaps.
- During testing, test reporting always takes place in a language (risks) that all stake-holder understands.
- Testing always concentrate on the most important matters first with optimal test delivery, in case of – limited time, money and qualified resources. With the time and resources we have, we just can able to complete 100% testing, so we need to determine a better way to

accelerate our testing effort with still managing the risk of the application under test. Efforts are not wasted on non-critical or low risk functions.

- Improve customer satisfaction – Due to customer involvement and good reporting and progress tracking.

Some Helpful Test Techniques to proceed Risk-based testing

- Path Flow testing
- Some amount of exploratory / Experience based testing
- Boundary Value analysis
- Equivalence partitioning
- Decision tables

REGRESSION TESTING?

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.

Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.

This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the new code changes are done.

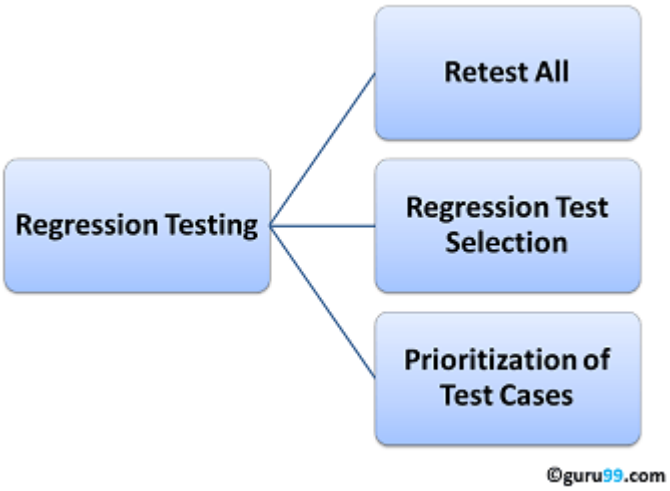
Need of Regression Testing

Regression Testing is required when there is a

- Change in requirements and code is modified according to the requirement
- New feature is added to the software
- Defect fixing
- Performance issue fix

How to do Regression Testing

Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of existing features. These modifications may cause the system to work incorrectly. Therefore, Regression Testing becomes necessary. Regression Testing can be carried out using the following techniques:



Retest All

- This is one of the methods for Regression Testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.

Regression Test Selection

- Instead of re-executing the entire test suite, it is better to select part of the test suite to be run
- Test cases selected can be categorized as 1) Reusable Test Cases 2) Obsolete Test Cases.
- Re-usable Test cases can be used in succeeding regression cycles.
- Obsolete Test Cases can't be used in succeeding cycles.

Prioritization of Test Cases

- Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

Selecting test cases for regression testing

It was found from industry data that a good number of the defects reported by customers were due to last minute bug fixes creating side effects and hence selecting the Test Case for regression testing is an art and not that easy. Effective Regression Tests can be done by selecting the following test cases -

- Test cases which have frequent defects
- Functionalities which are more visible to the users
- Test cases which verify core features of the product
- Test cases of Functionalities which has undergone more and recent changes
- All Integration Test Cases
- All Complex Test Cases
- Boundary value test cases
- A sample of Successful test cases
- A sample of Failure test cases

Regression Testing Tools

If your software undergoes frequent changes, regression testing costs will escalate.

In such cases, Manual execution of test cases increases test execution time as well as costs.

Automation of regression test cases is the smart choice in such cases.

The extent of automation depends on the number of test cases that remain re-usable for successive regression cycles.

Following are the most important tools used for both functional and regression testing in software engineering.

Ranorex Studio — all-in-one regression test automation for desktop, web, and mobile apps with built-in Selenium WebDriver. Includes a full IDE plus tools for codeless automation.

Testim - is an AI based test automation platform for web and mobile apps. It uses dynamic locators that learn with every execution and adapts to code changes, minimizing maintenance time spent fixing flaky tests

Selenium: This is an open source tool used for automating web applications. Selenium can be used for browser-based regression testing.

Quick Test Professional (QTP): HP Quick Test Professional is automated software designed to automate functional and regression test cases. It uses VBScript language for automation. It is a Data-driven, Keyword based tool.

Rational Functional Tester (RFT): IBM's rational functional tester is a Java tool used to automate the test cases of software applications. This is primarily used for automating regression test cases and it also integrates with Rational Test Manager.

Regression Testing and Configuration Management

Configuration Management during Regression Testing becomes imperative in Agile Environments where a code is being continuously modified. To ensure effective regression tests, observe the following :

- Code being regression tested should be under a configuration management tool
- No changes must be allowed to code, during the regression test phase. Regression test code must be kept immune to developer changes.
- The database used for regression testing must be isolated. No database changes must be allowed

Difference between Re-Testing and Regression Testing:

Retesting means testing the functionality or bug again to ensure the code is fixed. If it is not fixed,Defect needs to be re-opened. If fixed, Defect is closed.

Regression testing means testing your software application when it undergoes a code change to ensure that the new code has not affected other parts of the software.

STRUCTURED WALKTHROUGH?

A structured walkthrough, a static testing technique performed in an organized manner between a group of peers to review and discuss the technical aspects of software development process. The main objective in a structured walkthrough is to find defects inorder to improve the quality of the product.

Structured walkthroughs are usually NOT used for technical discussions or to discuss the solutions for the issues found. As explained, the aim is to detect error and not to correct errors. When the walkthrough is finished, the author of the output is responsible for fixing the issues.

Benefits:

- Saves time and money as defects are found and rectified very early in the lifecycle.
- This provides value-added comments from reviewers with different technical backgrounds and experience.
- It notifies the project management team about the progress of the development process.
- It creates awareness about different development or maintenance methodologies which can provide a professional growth to participants.

Structured Walkthrough Participants:

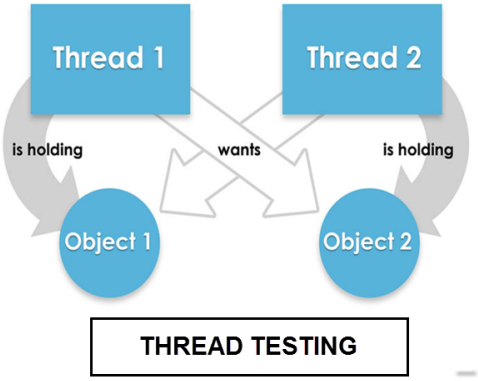
- **Author -** The Author of the document under review.
- **Presenter -** The presenter usually develops the agenda for the walkthrough and presents the output being reviewed.
- **Moderator -** The moderator facilitates the walkthrough session, ensures the walkthrough agenda is followed, and encourages all the reviewers to participate.
- **Reviewers -** The reviewers evaluate the document under test to determine if it is technically accurate.

- **Scribe -** The scribe is the recorder of the structured walkthrough outcomes who records the issues identified and any other technical comments, suggestions, and unresolved questions.

THREAD TESTING?

Thread testing is defined as a software testing type, which verify the key functional capabilities of a specific task(thread). It is usually conducted at the early stage of Integration Testing phase.

Thread based testing is one of the incremental strategies adopted during System Integration Testing. That's why, thread test should probably more properly be called a "**thread interaction test.**"



Types of Thread Testing

Thread based testing are classified into two categories

- **Single thread testing**: A single thread testing involves one application transaction at a time
- **Multi-thread testing**: A multi-thread testing involves several concurrently active transaction at a time

How to do Thread Testing

The thread process focuses on the integration activities rather than the full development lifecycle. For Example,

- Thread-based testing is a generalized form of session-based testing, in that sessions are a form of thread, but a thread is not necessarily a session.
- For thread testing, the thread or program (small functionality) are integrated and tested incrementally as a subsystem, and then executed for a whole system.
- At the lowest level, it provided integrators with better knowledge of the scope of what to test
- Rather than testing software components directly, it required integrators to concentrate on testing logical execution paths in the context of the entire system.

Tips for Multithread Testing

- Test your multithreaded program by executing it repeatedly with a different mix of applications running
- Test your multithreaded program by having multiple instances of the program active at the same time
- Execute your multithreaded program on different hardware models with varying stress levels and workloads
- Code inspection
- Only collect errors and failures that occurred in threads other than the main one

Disadvantages of Thread Testing

- For multithreading testing, the biggest challenge is that you should be able to program reproducible test for unit test
- Writing unit tests for multithreaded code is a challenging task
- Testing criteria for multi-thread testing are different than single thread testing. For multithread testing various factors like memory size, storage capacity, timing problems, etc. varies when called on different hardware.

What is Performance Testing?

- Performance Testing is defined as a type of software testing to ensure software applications will perform well under their expected workload.

Features and Functionality supported by a software system is not the only concern. A software application's performance like its response time, reliability, resource usage and scalability do matter. The goal of Performance Testing is not to find bugs but to eliminate performance bottlenecks.

The focus of Performance Testing is checking a software program's

- Speed - Determines whether the application responds quickly
- Scalability - Determines maximum user load the software application can handle.
- Stability - Determines if the application is stable under varying loads

Performance Testing is popularly called “Perf Testing” and is a subset of performance engineering.

Types of Performance Testing

- **Load testing -** checks the application's ability to perform under anticipated user loads. The objective is to identify performance bottlenecks before the software application goes live.
- **Stress testing -** involves testing an application under extreme workloads to see how it handles high traffic or data processing. The objective is to identify the breaking point of an application.
- **Endurance testing -** is done to make sure the software can handle the expected load over a long period of time.
- **Spike testing -** tests the software's reaction to sudden large spikes in the load generated by users.

- **Volume testing** - Under Volume Testing large no. of. Data is populated in a database and the overall software system's behavior is monitored. The objective is to check software application's performance under varying database volumes.
- **Scalability testing** - The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load. It helps plan capacity addition to your software system.

Common Performance Problems

Most performance problems revolve around speed, response time, load time and poor scalability. Speed is often one of the most important attributes of an application. A slow running application will lose potential users. Performance testing is done to make sure an app runs fast enough to keep a user's attention and interest. Take a look at the following list of common performance problems and notice how speed is a common factor in many of them:

- **Long Load time -** Load time is normally the initial time it takes an application to start. This should generally be kept to a minimum. While some applications are impossible to make load in under a minute, Load time should be kept under a few seconds if possible.
- **Poor response time -** Response time is the time it takes from when a user inputs data into the application until the application outputs a response to that input. Generally, this should be very quick. Again if a user has to wait too long, they lose interest.
- **Poor scalability -** A software product suffers from poor scalability when it cannot handle the expected number of users or when it does not accommodate a wide enough range of users. Load Testing should be done to be certain the application can handle the anticipated number of users.
- **Bottlenecking -** Bottlenecks are obstructions in a system which degrade overall system performance. Bottlenecking is when either coding errors or hardware issues cause a decrease of throughput under certain loads. Bottlenecking is often caused by one faulty section of code. The key to fixing a bottlenecking issue is to find the section of code that is causing the slowdown and try to fix it there. Bottlenecking is generally fixed by either fixing poor running processes or adding additional Hardware. Some **common performance bottlenecks** are

- CPU utilization
- Memory utilization
- Network utilization
- Operating System limitations
- Disk usage

Performance Testing Process

The methodology adopted for performance testing can vary widely but the objective for performance tests remain the same. It can help demonstrate that your software system meets certain pre-defined performance criteria. Or it can help compare the performance of two software systems. It can also help identify parts of your software system which degrade its performance.

Below is a generic process on how to perform performance testing

1. **Identify your testing environment -** Know your physical test environment, production environment and what testing tools are available. Understand details of the hardware, software and network configurations used during testing before you begin the testing process. It will help testers create more efficient tests. It will also help identify possible challenges that testers may encounter during the performance testing procedures.
2. **Identify the performance acceptance criteria -** This includes goals and constraints for throughput, response times and resource allocation. It is also necessary to identify project success criteria outside of these goals and constraints. Testers should be empowered to set performance criteria and goals because often the project specifications will not include a wide enough variety of performance benchmarks. Sometimes there may be none at all. When possible finding a similar application to compare to is a good way to set performance goals.

3. **Plan & design performance tests -** Determine how usage is likely to vary amongst end users and identify key scenarios to test for all possible use cases. It is necessary to simulate a variety of end users, plan performance test data and outline what metrics will be gathered.
4. **Configuring the test environment -** Prepare the testing environment before execution. Also, arrange tools and other resources.
5. **Implement test design -** Create the performance tests according to your test design.
6. **Run the tests -** Execute and monitor the tests.
7. **Analyze, tune and retest** - Consolidate, analyze and share test results. Then fine tune and test again to see if there is an improvement or decrease in performance. Since improvements generally grow smaller with each retest, stop when bottlenecking is caused by the CPU. Then you may have the consider option of increasing CPU power.

Performance Test Tools

There are a wide variety of performance testing tools available in the market. The tool you choose for testing will depend on many factors such as types of the protocol supported, license cost, hardware requirements, platform support etc. Below is a list of popularly used testing tools.

- LoadNinja – is revolutionizing the way we load test. This cloud-based load testing tool empowers teams to record & instantly playback comprehensive load tests, without complex dynamic correlation & run these load tests in real browsers at scale. Teams are able to increase test coverage. & cut load testing time by over 60%.
- NeoLoad - is the performance testing platform designed for DevOps that seamlessly integrates into your existing Continuous Delivery pipeline. With NeoLoad, teams test 10x faster than with traditional tools to meet the new level of requirements across the full Agile software development lifecycle - from component to full system-wide load tests.
- HP LoadRunner - is the most popular performance testing tools on the market today. This tool is capable of simulating hundreds of thousands of users, putting applications under real-life loads to determine their behavior under expected loads. Loadrunner features a virtual user generator which simulates the actions of live human users.
- Jmeter - one of the leading tools used for load testing of web and application servers.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

School of Computing

Department of Computer Science and Engineering

and

Department of Information Technology

SCS1608 - Software Quality Assurance and Testing

UNIT - V

UNIT – V

TAXANOMY TO TESTING TOOLS

A wide variety of software testing tools are available to cater to the different types of software, different programming languages, and to carry out different types of testing. These testing tools can be broadly divided into the following categories:

- Functional/Regression testing tools
- Source code testing tools
- Performance testing tools
- Java testing tools
- Embedded software testing tools
- Network protocol testing tools
- Configuration management/bug tracking tools
- Testing management tools

Functional/Regression Testing Tools:

These tools are used to test the application software and web applications such as web sites. As majority of the applications involve Graphical User Interface (GUI), the tools test the GUI objects and functionality automatically. These tools carry out black box testing. Client/Server applications, Enterprise Resource Planning (EPvP) software packages such as SAP,Customer Relations Management (CRM) software packages such as Siebel, web sites etc. can be tested for functionality using these tools. Whenever a change is made to the software, the software needs to be retested and hence these tools are also called regression testing tools. Compuware's QACenter, Segue Software's SilkTest, IBM Rational's Robot, Mecury Interactive's WinRunner belong to this category.

Source Code Testing Tools:

These tools check the source code of the application software. The testing is white box testing and hence the implementation details are taken into consideration. A number of tools are available for checking line coverage, branch coverage, and path coverage. For instance, the profilers display the number of times each line is executed. The test engineer can study the profiler output to find out which portions of the code are not executed and then create test cases in such a way that the lines, which were not executed earlier, can be executed. Tools are also available to test whether the source code is compliant to the standard

coding guidelines and to generate the metrics such as number of non-commented lines, number of commented lines, number of functions etc. Some tools check the portability of the code. For example, the code written in C is not portable if operating system dependent features are used. The 'lint' utility in Unix/Linux systems checks the portability of C code. Some application software source code testing tools are: AutomatedQA's AQtime, Parasoft's Insure++, and Telelogic's Logiscope.

Performance Testing Tools:

These tools are used to carry out performance testing or stress testing. These tools are very useful to test how the application works when multiple users access the application simultaneously. The application can be for example, a database or a web site. These tools simulate multiple users on a single machine and hence you do not need many machines and many test engineers to do the performance testing. AutoTester's AutoController, Compuware's QALoad, Mercury Interactive's LoadRunner, Segue Software's SilkPerformer, IBM Rational's Performance Tester, Apache JMeter belong to this category. Some other specialized tools in this category are Argogroup's MonitorMaster for testing mobile applications that use WML and XHTML, Short Messaging Service (SMS) and Multimedia Messaging Service (MMS); IBM Rational's Prevue-X and Prevue-ASCII for X-windows and ASCII terminal emulators.

Java Testing Tools:

As Java has become a popular programming language in recent years, a number of tools are available exclusively for testing Java applications. These tools are for testing applications written in Java programming language and for testing Java classes. Jemmy is an open source library to create automated tests for Java GUI applications. JMeter of Apache is another open source software to do performance testing. Parasoft's jtest is used for Java class testing.

Embedded Software Testing tools:

Testing embedded software is a very challenging task as the timing requirements for these applications are very stringent. In embedded systems, the code has to be optimized so that it occupies the minimum memory. IBM Rational Test Real Time is the widely used test tool in this category.

Network Protocol Testing tools:

As computer networks are becoming widespread, testing networking/communication software has attained lot of importance in recent years. A number of tools are available for testing networking and communication protocols. Many test instrumentation vendors such as Agilent Technologies, Rhode & Schwartz etc. supply protocol analyzers which generate the necessary protocols based on international standards such as ITU-T standards. netlQ's ANVL (Automated Network Validation Library) is to test routers and other networking products. This software generates packets in correct and incorrect formats to test the networking software. netlQ's Chariot is a network performance testing tool.

Configuration Management/Bug Tracking Tools:

In large software development projects, configuration management is a very important process. Also, when the test engineers report bugs, the managers have to track these bugs and ensure that all the bugs are removed. To facilitate this activity, good workflow management software is important. Many such tools, which are web-based are available. Bugzilla's Bugzilla is an open source defect tracking system. Samba's Jitterbug is a freeware defect tracking system. IBM Rational Software's Clear DDTS is a change request management software. GNU's GNATS is a freeware bug tracking and change management software. Segue Software's SilkRadar is an issue tracking and bug tracking software. Microsoft Excel is also used extensively for bug tracking. In Unix/Linux/ Solaris systems, utilities are available for version and release control of source code— these utilities are Source Code Control System (SCCS) and Revision Control System (RCS).

Software Testing Management Tools:

These tools help in managing process-oriented software testing. Using these tools, the QA manager can create a formal test plan, allocate resources, schedule unattended testing, track the status of various bugs/ activities etc. Auto Tester's AutoAdviser, Computer Software's QADirector, QIS's QCIT, Segue Software's SilkPlan Pro, IBM Rational Test Manager, Mercury Interactive's TestDirector are some such tools.

TEST AUTOMATION TOOL EVALUATION

1.Introduction

Success in any Test Automation (TA) effort lies in identifying the right tool for automation. A detailed analysis of various tools must be performed before selecting a tool. This requires a lot of effort and planning. The effort and learning

obtained during tool evaluation will in turn help during the execution of the TA project.

Many companies fall prey to the sales executives of tool vendors, who show how easy it is to create scripts using their tool. It therefore becomes necessary to go into the details of the script thus created, to check the way by which the tool identifies and works with the product. In many cases, organizations have bought a tool license because it worked fine using record/playback. Eventually, they find out that the tool worked by identifying the product and its controls using coordinate positions, which is not a very reliable method, and the tool gets shelved. Hence a systematic approach must be taken to evaluate tools.

The entire process of tool evaluation can be broken down into three major phases:

- Requirements Gathering
- Tool Selection
- POC using selected tool

2.Requirements Gathering

During the requirements gathering phase of tool evaluation one has to list out the requirements for the automation tool. Some of the important questions you will need to ask would be:

- What problems will the tool solve?
- What technical capabilities will the tool need, to be compatible with your environment?

Some of the items that will help you guide your requirements list:

1. Compatibility issues

2 Tool audience

3 Management goals

4 Testing requirements

5 Technology

Compatibility Issues

Your testing tool will need to be compatible with:

- The operating systems your product supports
- The development environments used to create your product
- Third party software with which your product integrates
- The test management tool used (in order to be able to integrate with it); this will eliminate data redundancy and will give the advantage of managing all test related data at a single location
- The tool should also be version control friendly so that scripts created can be brought under source code control

Tool Audience

The skills of people involved in test automation and that of the people who use the automation scripts is another important criterion for the test tool evaluation process. The benefits obtained through automation boils down to how effectively the tool is being put to use.

Will your organization allow for staff training? Can your organization, within the implementation time, allow for the learning curve required to become comfortable with the tool?

Management goals

Typically, this will be based on the product roadmap and the goal for automation from the management perspective. Based on the product roadmap, the management might consider reviewing the time for which the tool has been in the market. They may also consider whether the tool will be upgraded periodically to support newer technologies. Any management will have a budget and will fit efforts within a given budget. So it is also important for the management to consider the licensing and maintenance cost of a tool and the additional hardware required for running the scripts.

Testing Requirements

What type of testing problems do you want the testing tool to address?

- Manual testing problems
- Time constraints when implementing small changes in the system
- Shorter regression testing timeframes
- Test data setup
- Defect tracking
- Increased test coverage
- Increased efficiency of the testing process

Technology

When considering technology requirements in the automation perspective, there are two views that must be taken:

- Technologies that should be supported by the tool
- Features required in the tool

The requirement could be for the tool to support all technologies that are used in the product and any new technology that is being planned to be implemented in the future. A list of all the technologies used in the product and the platforms/browsers on which the product is supported must be created as the technical requirements for the tool. For a tool to be able to automate testing of the product, the important criterion is the tool's ability to identify, access and work with all controls used in the application. Hence it is important to create a list of all controls (standard, custom and third party) used in the application and check if the tool is able to identify and handle all the listed controls.

The technical stakeholders of the automation project will have a list of desirable features that they need in the tool. This list of desirable features must be created in consultation with the technical team. Apart from this, a list of all features supported by the tool must also be created so that all features of the tool are considered during evaluation. An example of few features that can be listed are the tool's support for testing Graphical User Interface (GUI), Application Programming Interface (API) and Command Line Interface (CLI), support to extend tool using Open APIs like Win32 APIs, verification points supported in the tool, support of an efficient Recovery System and support for test report creation.

At the end of the requirements gathering phase, all necessary points to consider for selecting an automation tool are available. These points form the evaluation criteria for the tool.

1.Tools Selection

The second step in test automation tool evaluation is the selection of tools. While selecting tools it is important to remember that no single tool will satisfy all the requirements. The tool that meets most of the evaluation criteria should be chosen after discussion with stakeholders. Based on the tools limitations with respect to requirements, the automation activities must be planned.

All tools that meet most of the evaluation criteria can considered for evaluation. When many tools are found to satisfy the evaluation criteria, further analysis of

tools should be done. Do a feature categorization by listing each tool according to the following features it provides:

- Mandatory features: These are the features that are essential to accomplish your goal in meeting your requirements within the constraints

- Desirable features: these are features that will distinguish the best tools from the others

- Irrelevant features: Features that are not important and will not provide any real benefit to your situation.

Rate these features and assess as many tools as possible to prepare a short list of a few tools. Contact the relevant vendors and possibly ask for an evaluation version to run your proof of concept (POC).

2.POC Using Selected Tool

The last phase of tool evaluation is doing a proof of concept. Though conceptually the tool appears to satisfy the evaluation criteria, it is necessary to try the tool for a few test scenarios / cases in the product. Every tool vendor provides an evaluation version of their tool for this purpose for a limited period of time. It is sufficient to use the evaluation version for the POC.

The scenarios chosen for POC are very important. They should be chosen in such a way that the scenarios cover most of the controls and a few common features present across the product. It should be a sample,

which, when automated should give the confidence that automation using the tool for the product will be successful. This will also help in finding available resources' competence in using the tool. In case the POC fails, the reasons for failure must be analyzed and documented. The next tool for POC should be

selected and this can be an iterative process till you identify the tool that most likely,suits your requirements.

Conclusion

Tool evaluation is indeed a process in itself and requires a lot of research irrespective of who does the evaluation. The above listed process allows for you to make an informed decision with regards to the best tool to assist you with your software test automation effort.

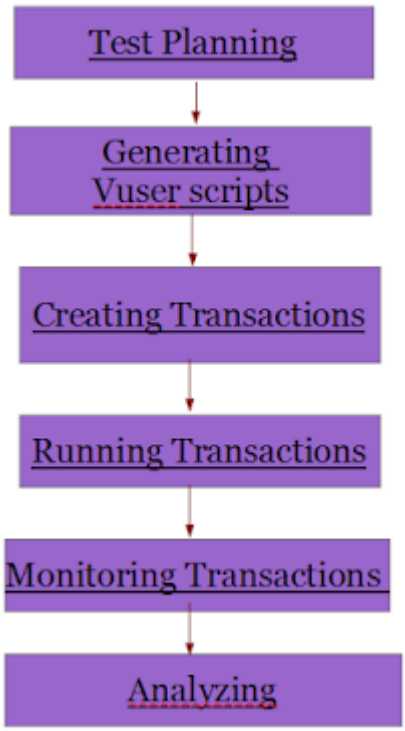
LOAD RUNNER

Why Load Runner??:

1. Load Runner: One of the best automated performance testing tool.
2. Uses ANSI C as the default programming language and other languages like Java and VB.
3. No need to install it on the server under test. It uses native monitors.
4. Supports all types of protocols (HTTP, FTP and SMTP).
5. Easy to analyze the results and creating scripts.

As we know Load Runner was acquired by **Hewlett-Packard** organization. One of the best and most used tool by many organizations. Though it's a paid tool, for perfect and easiness in use this tool is the best.

For every tool, there is a testing process to test an application. For the Load Runner , the best process to follow is :



The above model is the best model to follow for the performance test. Let's discuss much about the tool now. There are main components of Load Runner. They' are:

Components of Load Runner:

1) Vuser Generator
2) Controller
3)Analyzer

Let's discuss about each component briefly with related examples.

VUser Generator:

The first component and the basic component is "Vuser Generator". In Load Runner tool, humans are replaced by Vusers who are the replica of humans. More number of Vusers can be worked on a single work station with different scenarios. Load runner can accommodate hundreds or even thousands of Vusers with different scenarios.

With the help of Vuser script, users can perform the tests. User can record and playback the application for script generation.By modification or editing the scripts, user can create different scenarios for different Vusers. With this load test can be made simple and easy with one workstation.

Load runner supports scripting languages like ANSI C, VB Script, JAVA etc.. C and VB scripting are the most used ones in load runner. In recent versions of load runner JAVA scripting has been implemented, which can be widely used.

Controller:

In Load Runner 'Controller' is used to control the VUsers with single work station with different scenarios assigned to VUsers.

Analysis:

After the performance test the user can view the results of the test in graphs.More into the

concepts of Load Runner. As we discussed, we start with 'Vuser Generator'.

Vuser scripts are created by Virtual User Generator with the recording of activities between client and server. It records the scripts. These scripts are used to emulate the steps of real human users. Using Vugen, we can also run the scripts for debugging.
VuGen can be used for recording in windows platforms. But, a recorded Vuser script can also be run on Unix platform.

Developing Vuser Script is a five step process:

- Record a Vuser script
- Vuser Script Enhancement – by adding the control statements and other functions
- Run time Settings Configuration
- Running of Vuser Script on Stand Alone machine – Verify that the script runs correctly
- Integration of Vuser Script – into a LoadRunner scenario or Performance Center or Tuning module session or Business process monitor profile etc.

WINRUNNER

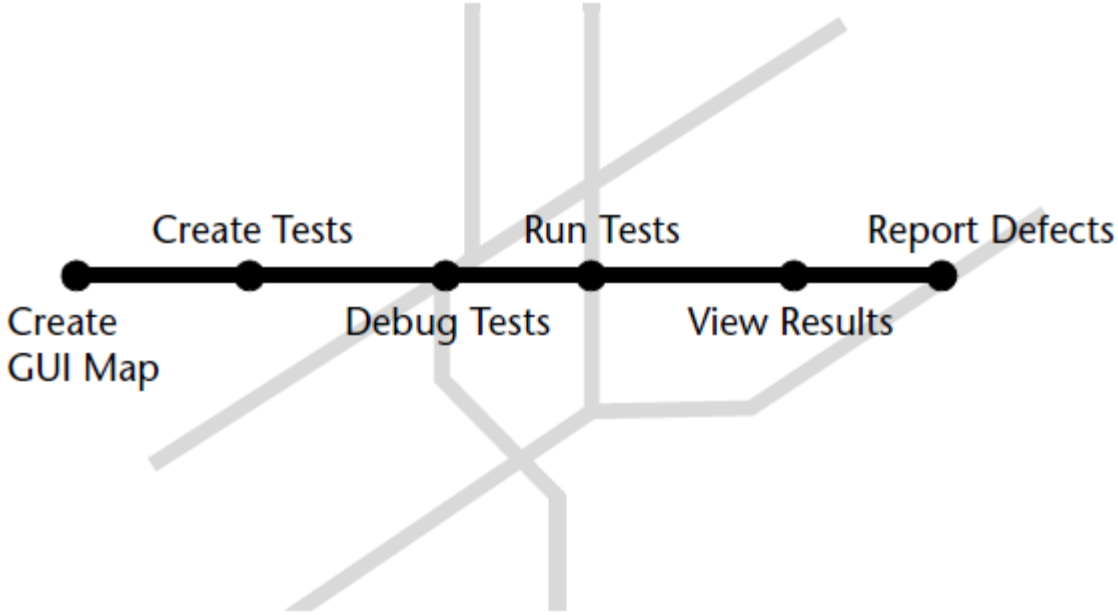
WinRunner is a testing tool to do functional/regression testing. Using WinRunner, you can record GUI operations. While recording, WinRunner automatically creates a test script. This test script can be run automatically later on for carrying out unattended testing. The important aspects of WinRunner are:

- You can do functional/regression testing of a variety of application software written in programming languages such as PowerBuilder, Visual Basic, C/C++, and Java. You can also carry out the testing on ERP/CRM software packages.
- You can do the testing in all flavors of Windows operating systems and different browser environments such as Internet Explorer and Netscape Navigator.

- You can record the GUI operations in the 'record' mode. WinRunner automatically creates a tesi script. This test can be modified if required and can be executed later on in unattend*aA*mode. The recovery manager enables the application to be brought to a known state in case there is a problem during the unattended testing. Rapid Test Script Wizard creates the test scripts automatically.
- You can add checkpoints to compare actual and expected results. The checkpoints can be GUI checkpoints, bitmap checkpoints and web links.
- It provides a facility for synchronization of test cases.
- Data Driver Wizard provides the facility to convert a recorded test into a data driven test. So, you c*i replace data with variables within a test script. For example, you can test a login process by taking the input for username and password fields from a database.
- Database checkpoints are used to verify data in a database during automated testing. The records that are inserted, deleted, modified, or updated will be highlighted so that you can ensure database integrity and transaction accuracy.
- The Virtual Object Wizard of WinRunner is used to teach WinRunner to recognize, record, and replay custom objects.
- The reporting tools provide the facility to generate automatically the test reports and analyze the defects.
- WinRunner can be integrated with the testing management tool TestDirector to automate many of the activities in the testing process

Testing with WinRunner involves six main stages:

Testing with *WinRunner* involves six main stages:



Create the GUI Map

The first stage is to create the GUI map so WinRunner can recognize the GUI objects in the application being tested. Use the RapidTest Script wizard to review the user interface of your application and systematically add descriptions of every GUI object to the GUI map. Alternatively, you can add descriptions of individual objects to the GUI map by clicking objects while recording a test.

Note that when you work in *GUI Map per Test* mode, you can skip this step.

Create Tests

Next, you create test scripts by recording, programming, or a combination of both. While recording tests, insert checkpoints where you want to check the response of the application being tested. You can insert checkpoints that check GUI objects, bitmaps, and databases. During this process, WinRunner captures data and saves it as expected results—the expected response of the application being tested.

Debug Tests

You run tests in Debug mode to make sure they run smoothly. You can set breakpoints, monitor variables, and control how tests are run to identify and isolate defects. Test results are saved in the debug folder, which you can discard once you've finished debugging the test.

When WinRunner runs a test, it checks each script line for basic syntax errors, like incorrect syntax or missing elements in ***If***, ***While***, ***Switch***, and ***For*** statements. You can use the Syntax Check options ***Tools Syntax Check) to check for these types of syntax errors before running your test.***

Run Tests

You run tests in Verify mode to test your application. Each time WinRunner encounters a checkpoint in the test script, it compares the current data of the application being tested to the expected data captured earlier. If any mismatches are found, WinRunner captures them as actual results.

View Results

You determine the success or failure of the tests. Following each test run, WinRunner displays the results in a report. The report details all the major events that occurred during the run, such as checkpoints, error messages, system messages, or user messages.

If mismatches are detected at checkpoints during the test run, you can view the expected results and the actual results from the Test Results window. In cases of bitmap mismatches, you can also view a bitmap that displays only the difference between the expected and actual results.

You can view your results in the standard WinRunner report view or in the Unified report view. The WinRunner report view displays the test results in a Windows-style viewer. The Unified report view displays the results in an HTML-style viewer (identical to the style used for QuickTest Professional testresults).

Report Defects

If a test run fails due to a defect in the application being tested, you can report information about the defect directly from the Test Results window.

This information is sent via e-mail to the quality assurance manager, who tracks the defect until it is fixed.

You can also insert **tddb_add_defect** statements to your test script that instruct WinRunner to add a defect to a TestDirector project based on conditions you define in your test script.

Rational Funtional Testing Tool

Rational Functional Tester is a tool for automated testing of software applications from the Rational Software division of IBM. It allows users to create tests that mimic the actions and assessments of a human tester.[1] It is primarily used by Software Quality Assurance teams to perform automated regression testing.

Rational Functional Tester is a software test automation tool used by quality assurance teams to perform automated regression testing. Testers create scripts by using a test recorder which captures a user's actions against their application under test. The recording mechanism creates a test script from the actions. The test script is produced as either a Java or Visual Basic.net application, and with the release of version 8.1, is represented as series of screen shots that form a visual storyboard. Testers can edit the script using standard commands and syntax of these languages, or by acting against the screen shots in the storyboard. Test scripts can then be executed by Rational Functional Tester to validate application functionality. Typically, test scripts are run in a batch mode where several scripts are grouped together and run unattended.

During the recording phase, the user may introduce verification points, which capture an expected system state, such as a specific value in a field, or a given property of an object, such as enabled or disabled. During playback, any discrepancies between the baseline captured during recording and the actual result achieved during playback are noted in the Rational Functional Tester log. The tester can then review the log to determine if an actual software bug was discovered.

Key Technologies

Storyboard Testing

Introduced in version 8.1 of Rational Functional Tester, this technology enables testers to edit

test scripts by acting against screen shots of the application.

Object

The Rational Functional Tester Object Map is the underlying technology used by Rational Functional Tester to find and act against the objects within an application. The Object Map is automatically created by the test recorder when tests are created and contains a list of properties used to identify objects during playback.

ScriptAssure

During playback, Rational Functional Tester uses the Object Map to find and act against the application interface. However, during development it is often the case that objects change between the time the script was recorded and when a script was executed. ScriptAssure technology enables Rational Functional Tester to ignore discrepancies between object definitions captured during recording and playback to ensure that test script execution runs uninterrupted. ScriptAssure sensitivity, which determines how big an object map discrepancy is acceptable, is set by the user.

Data Driven Testing

It is common for a single functional regression test to be executed multiple times with different data. To facilitate this, the test recorder can automatically parametrize data entry values, and store the data in a spreadsheet like data pool. This enables tester to add additional test data cases to the test data pool without having to modify any test code. This strategy increases test coverage and the value of a given functional test.

Dynamic Scripting Using Find API

Rational Functional Test script, Eclipse Integration uses Java as its scripting language. The Script is a .java file and has full access to the standard Java APIs or any other API exposed through other class libraries.

Apart from this RFT itself provides a rich API to help user further modify the script generated through the recorder. RationalTestScript class that is the base class for any TestScript provides a find API that can be used to find the control based on the given properties.

Domains supported

(list is made based on the information for v 8.5, see here)

HTML Support: Mozilla Firefox, Internet Explorer, Google Chrome

Java

Abstract Window Toolkit (AWT) controls

Standard Widget Toolkit (SWT) controls

Swing or Java Foundation Class (JFC) controls

Eclipse (32-bit and 64-bit)

Dojo

WinForm and Windows Presentation Framework based .NET applications

ARM

Ajax

SAP WebDynPro

SAP - SAPGUI

Siebel

Silverlight

GEF

Flex

PowerBuilder

Visual Basic

Adobe PDF

Functional Tester Extensions for Terminal-based applications

SILK TEST

Introduction

Silk Test is a tool specifically designed for doing **REGRESSION AND FUNCTIONALITY**testing. It is developed by Segue Software Inc.

Silk Test is the industry's leading functional testing product for e-business applications, whether Window based, Web, Java, or traditional client/server-based. Silk Test also offers test planning, management, direct database access and validation, the flexible and robust 4Test scripting language, a built in recovery system for unattended testing, and the ability to test across multiple platforms, browsers and technologies.

You have two ways to create automated tests using silktest:

1. Use the **Record Testcase command** to record actions and verification steps as you navigate through the application.

2. Write the testcase manually using the **Visual 4Test scripting language**.

1. **Record Testcase**

The Record / Testcase command is used to record actions and verification steps as you navigate through the application. Tests are recorded in an object-oriented language **called Visual 4Test**. The recorded testreads like a logical trace of all of the steps that were completed by the user. The Silk Test point and click verification system allows you to record the verification step by selecting from a list of properties that are appropriate for the type of object being tested. For example, you can verify the text is stored in a text field.

2. **Write the Testcase manually**

We can write tests that are capable of accomplishing many variations on a test. The key here is re-use. A test case can be designed to take parameters including input data and expected results. This "data-driven" testcase is really an instance of a class of test cases that performs certain steps to drive and verify the application-under-test. Each instance varies by the data that it carries.

Since far fewer tests are written with this approach, changes in the GUI will result in reduced effort in updating tests. A data-driven test design also allows for the externalization of testcase data and makes it possible to divide the responsibilities for developing testing requirements and for developing test automation. For example, it may be that a group of domain experts create the Testplan Detail while another group of test engineers develop tests to satisfy those requirements.

In a script file, an automated testcase ideally addresses one test requirement. Specifically, a 4Test function that begins with the test case keyword and contains a sequence of 4Test statements. It drives an application to the state to be tested, verifies that the application works as expected, and returns the application to its base state.

A script file is a file that contains one or more related testcases. A script file has a .t extension, such as find .t

TESTING TOOLS FOR JAVA

1. Arquillian

Arquillian is a highly innovative and extendible testing platform for JVM that allows developers to easily create automated integration, functional and acceptance tests for Java. Arquillian allows you to run test in the run-time so you don't have to manage the run-time from the test (or the build). Arquillian can be used to manage the life cycle of the container (or containers),bundling test cases, dependent classes and resources. It is also capable of deploying archive into containers and execute tests in the containers and capture results and create reports.

Arquillian integrates with familiar testing frameworks such as JUnit 4, TestNG 5 and allows tests to be launched using existing IDE, and because of its modular design it is capable of running Ant and Maven test plugins.

2. JTest

JTest also known as ‘Parasoft JTest’ is an automated Java software testing and static analysis software made by Parasoft. JTest includes functionality for Unit test-case generation and execution, static code analysis, data flow static analysis, and metrics analysis, regression testing, run-time error detection.

There are also features that allow you to peer code review process automation and run-time error detection for e.g.: Race conditions, exceptions, resource and memory leaks, security attack vulnerabilities.

3. The Grinder

‘The Grinder’ is a Java load testing framework that was designed to make sure it was easy to run and distributed test’s using many load injector machines. The Grinder can Load test on anything that has a Java API. This includes HTTP web servers, SOAP and REST web services, and application servers and including custom protocols and the test scripts are written in the powerful Jython and Clojure languages. The GUI console for The Grinder allows you to have multiple load injectors to be monitored and controlled and Automatic management of client connections and cookies, SSL, Proxy aware and Connection throttling.

It is freely available under a BSD-style open-source license. You can find out more on their website.

4. TestNG

TestNG is a testing framework designed for the Java programming language and inspired by JUnit and NUnit. TestNG was primarily designed to cover a wider range of test categories such as unit, functional, end-to-end, integration, etc. It also introduced some new functionality that make it more powerful and easier to use, such as: Annotations, Running tests in big thread pools with various policies available, code testing in a multi thread safe, flexible test configurations, data-driven testing support for parameters, and more.

TestNG is supported by a variety of tools and plug-ins such as Eclipse, IDEA, Maven, etc

5. JUnit

JUnit is a unit testing framework designed for the Java programming language. JUnit has played an important role in the development of test-driven development frameworks. It is one of a family of unit testing frameworks which is collectively known as the xUnit that originated with SUnit.

JUnit is linked as a JAR at compile-time and can be used to write repeatable tests

6. JWalk

JWalk is designed as a unit testing toolkit for the Java programming language. It has been designed to support a testing paradigm called Lazy Systematic Unit Testing. The JWalkTester tool performs any tests of any compiled Java class, supplied by a programmer. It is capable of testing conformance to a lazy specification, by static and dynamic analysis, and from hints by the programmer behind the code.

7. Mockito

Mockito is designed as a open source testing framework for Java which is available under a MIT License. Mockito allows programmers to create and test double objects (mock objects) in automated unit tests for the purpose of Test-driven Development (TDD) or Behavior Driven Development (BDD).

8. Powermock

PowerMock is a Java Framework for unit testing of source code and It runs as an extension of other Mocking frameworks like Mockito or EasyMock but comes with more powerful capabilities. PowerMock utilizes a custom classloader and bytecode manipulator to enable mocking of static methods, removal of static initializes, constructors, final classes and methods and private methods. It as been primarily designed to extend the existing API's with a small number of methods and annotations to enable the extra features.

It is available under an open source Apache License 2..

9. JMeter

JMeter is a software that can perform load test, performance-oriented business (functional) test, regression test, etc., on different protocols or technologies.

Stefano Mazzocchi of the Apache Software Foundation was the original developer of JMeter. He wrote it primarily to test the performance of Apache JServ (now called as Apache Tomcat project). Apache later redesigned JMeter to enhance the GUI and to add functional testing capabilities.

JMeter is a Java desktop application with a graphical interface that uses the Swing graphical API. It can therefore run on any environment / workstation that accepts a Java virtual machine, for example − Windows, Linux, Mac, etc.

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 Features

Following are some of the features of JMeter −

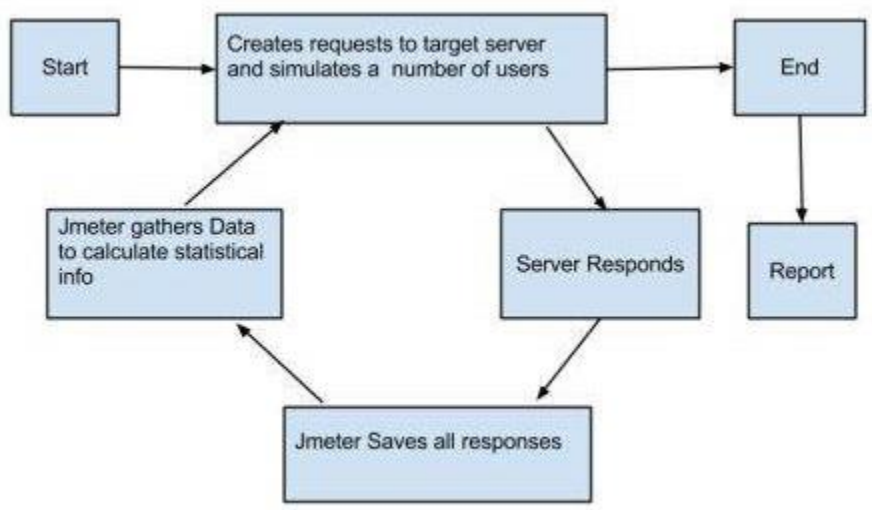
- Being an open source software, it is freely available.
- It has a simple and intuitive GUI.
- JMeter can conduct load and performance test for many different server types − Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3, etc.
- It is a platform-independent tool. On Linux/Unix, JMeter can be invoked by clicking on JMeter shell script. On Windows, it can be invoked by starting the jmeter.bat file.

- It has full Swing and lightweight component support (precompiled JAR uses packages javax.swing.*).
- JMeter store its test plans in XML format. This means you can generate a test plan using a text editor.
- Its full multi-threading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- It is highly extensible.
- It can also be used to perform automated and functional testing of the applications.

How JMeter Works?

JMeter simulates a group of users sending requests to a target server, and returns statistics that show the performance/functionality of the target server/application via tables, graphs, etc.

Take a look at the following figure that depicts how JMeter works −



What is Junit?

Junit is widely used testing framework along with Java Programming Language. You can use this automation framework for both unit testing and UI testing.It helps us define the flow of execution of our code with different Annotations. Junit is built on idea of "first testing and then coding" which helps us to increase productivity of test cases and stability of the code.

Important Features of Junit Testing -

1. It is open source testing framework allowing users to write and run test cases effectively.
2. Provides various types of annotations to identify test methods.
3. Provides different Types of Assertions to verify the results of test case execution.
4. It also gives test runners for running tests effectively.
5. It is very simple and hence saves time.
6. It provides ways to organize your test cases in form of test suits.
7. It gives test case results in simple and elegant way.
8. You can integrate Jnuit with Eclipse ,Android Studio,Maven & Ant

Following are the JUnit extensions −

- Cactus
- JWebUnit
- XMLUnit
- MockObject

Cactus

Cactus is a simple test framework for unit testing server-side java code (Servlets, EJBs, Tag Libs, Filters). The intent of Cactus is to lower the cost of writing tests for server-side code. It uses JUnit and extends it. Cactus implements an in-container strategy that executes the tests inside a container.

Cactus ecosystem is made of several components −

- **Cactus Framework** is the heart of Cactus. It is the engine that provides the API to write Cactus tests.

- **Cactus Integration Modules** are front-ends and frameworks that provide easy ways of using the Cactus Framework (Ant scripts, Eclipse plugin, and Maven plugin).