Chapter 1

Quality Concept

1.0 OBJECTIVES

Dear students, this chapter covers the overview of quality aspects and focuses on software quality metrices. After studying this chapter you will be able to-

- Discuss definition of Quality, QA, SQA and Quality factors
- Explain the Software Quality Metrics
- Explain Process Improvement
- Discuss Process and Product Quality
- Describe the SEI Process Capability Maturity model, ISO, Six-Sigma

1.1 INTRODUCTION

Competition to provide specialized products and services results in breakthroughs as well as long-term growth and change. Quality assurance verifies that any customer offering, regardless if it is new or evolved, is produced and offered with the best possible materials, in the most comprehensive way, with the highest standards. The goal to exceed customer expectations in a measurable and accountable process is provided by quality assurance.

1.2 DEFINITION OF QUALITY, QA, SQA

Quality:

The term 'quality' is often used in a vague, blurred way. If someone talks about 'working on quality', they may simply mean activities designed to improve the organisation and its services. Quality is essentially about learning what you are doing well and doing it better. It also means finding out what you may need to change to make sure you meet the needs of your service users. Quality is about:

- knowing what you want to do and how you want to do it
- learning from what you do
- using what you learn to develop your organisation and its services
- seeking to achieve continuous improvement
- Satisfying your stakeholders - those different people and groups with an interest in your organisation.

Quality assurance is the process of verifying or determining whether products or services meet or exceed customer expectations. Quality assurance is a process-driven approach with specific steps to help define and attain goals. This process considers design, development, production, and service.

The most popular tool used to determine quality assurance is the Shewhart Cycle, developed by Dr. W. Edwards Deming. This cycle for quality assurance consists of four steps: *Plan*, *Do*, *Check*, and *Act*. These steps are commonly abbreviated as PDCA.

The four quality assurance steps within the PDCA model stand for -

- **Plan:** Establish objectives and processes required to deliver the desired results.
- **Do:** Implement the process developed.
- Check: Monitor and evaluate the implemented process by testing the results against the predetermined objectives
- **Act:** Apply actions necessary for improvement if the results require changes.

PDCA is an effective method for monitoring quality assurance because it analyzes existing conditions and methods used to provide the product or service customers. The goal is to ensure that excellence is inherent in every component of the process. Quality assurance also helps determine whether the steps used to provide the product or service are appropriate for the time and conditions. In addition, if the PDCA cycle is repeated throughout the lifetime of the product or service, it helps improve internal company efficiency.

Quality assurance demands a degree of detail in order to be fully implemented at every step. *Planning*, for example, could include investigation into the quality of the raw materials used in manufacturing, the actual assembly, or the inspection processes used. The *Checking* step could include customer feedback, surveys, or other marketing vehicles to determine if customer needs are being exceeded and why they are or are not. *Acting* could mean a total revision in the manufacturing process in order to correct a technical or cosmetic flaw.

Quality Control : The terms “quality assurance” and “quality control” are often used interchangeably to refer to ways of ensuring the quality of a service or product. The terms, however, have different meanings.

- **Assurance:** The act of giving confidence, the state of being certain or the act of making certain.
- **Quality assurance:** The planned and systematic activities implemented in a quality system so that quality requirements for a product or service will be fulfilled.
- **Control:** An evaluation to indicate needed corrective responses; the act of guiding a process in which variability is attributable to a constant system of chance causes. Quality control: The observation techniques and activities used to fulfill requirements for quality.

Software Quality Assurance

Business software is never built overnight. It takes a lot of planning, consultation and testing to be able to come up with an initial version of the application. If a business hurries up the development of a certain application, they would end up spending more in addressing the problem the application brought than they have earned.

This could even be more frustrating when the software being developed is set to be sold to customers or for public use. But a bug free program is not the only goal of a company. If they follow that concept they would end up developing a very simple program without any special features and would frustrate their users and hinder their productivity.

That is where the concept of software quality assurance comes in. For most developers software quality assurance offers a comprehensive solution in ensuring that the application they are working on is functioning and lives up to expectations. Quality assurance could be found everywhere. Products in the market also undergo quality assurance to make sure the goods are made according to expectations. However, once the products are sent out to the market, they are already done and it is up to the customers on how to be satisfied with the product. On the other hand software quality assurance takes on a different side. Aside from delivering good products, software quality assurance or simply known as SQA follows a set protocols that are universally recognized. By using these protocols users are assured the application they are using

are built according to standards and every stage of software development follows the same standards.

SQA Objectives
Software Quality Assurance was created with the following objectives:
Small to Zero Defects After Installation – One of the biggest goals of SQA is to prevent any possible defects when the output is made. Developers and engineers have to use universally approved steps to ensure that the program was built up to expectations but also to prevent errors in the system. Although some standards allow as much as .04 errors in the system, zero-error is still the system's target. When there's zero-error, the program is more likely to have zero crash scenarios. The ability to handle stress of a program is different from the errors it has but crashes usually comes from defects so prevention of defects will most likely yield a continuously working application.

Customer Satisfaction – Everything else will be just nothing if the customers don't like what they see. Part of SQA is to ensure that software development was made according to their needs, wants and exceeding their expectations. Even if the bugs and errors are minimized by the system, customer satisfaction is more important and should be emphasized.

Well Structured – SQA takes care of the stages of application construction. Anyone could be easily build an application and launch it in their environment without any glitches. However, not everyone could easily build an application that could be understood well. SQA ensures that each application are build in an understandable manner. Their applications could easily be transferred from one developer to another.

1.3 QUALITY FACTORS

The various factors, which influence the software, are termed as software factors. They can be broadly divided into two categories. The classification is done on the basis of measurability. The first category of the factors is of those that can be measured directly such as number of logical errors and the second category clubs those factors which can be measured only indirectly for example maintainability but the each of the factors are to be measured to check for the content and the quality control. Few factors of quality are available and they are mentioned below.

- Correctness - extent to which a program satisfies its specification and fulfills the client's objective.
- Reliability - extent to which a program is supposed to perform its function with the required precision.
- Efficiency - amount of computing and code required by a program to perform its function.

- Integrity - extent to which access to software and data is denied to unauthorized users.
- Usability- labor required to understand, operate, prepare input and interpret output of a program
- Maintainability- effort required to locate and fix an error in a program.
- Flexibility- effort needed to modify an operational program.
- Testability- effort required to test the programs for their functionality.
- Portability- effort required to run the program from one platform to other or to different hardware.
- Reusability- extent to which the program or it's parts can be used as building blocks or as prototypes for other programs.
- Interoperability- effort required to couple one system to another.
  Now as you consider the above-mentioned factors it becomes very obvious that the measurements of all of them to some discrete value are quite an impossible task. Therefore, another method was evolved to measure out the quality. A set of matrices is defined and is used to develop expressions for each of the factors as per the following expression

Fq = C1*M1 + C2*M2 + ……Cn*Mn

Where Fq is the software quality factor, Cn are regression coefficients and Mn is metrics that influences the quality factor. Metrics used in this arrangement is mentioned below -

- Audit ability- ease with which the conformance to standards can be verified.
- Accuracy- precision of computations and control
- Communication commonality- degree to which standard interfaces, protocols and bandwidth are used.
- Completeness- degree to which full implementation of functionality required has been achieved.
- Conciseness- program's compactness in terms of lines of code.
- Consistency- use of uniform design and documentation techniques throughout the software development.
- Data commonality- use of standard data structures and types throughout the program.
- Error tolerance – damage done when program encounters an error.
- Execution efficiency- run-time performance of a program.
- Expandability- degree to which one can extend architectural, data and procedural design.
- Hardware independence- degree to which the software is de-coupled from its operating hardware.
- Instrumentation- degree to which the program monitors its own operation and identifies errors that do occur.

- Modularity- functional independence of program components.
- Operability- eases of programs operation.
- Security- control and protection of programs and database from the unauthorized users.
- Self-documentation- degree to which the source code provides meaningful documentation.
- Simplicity- degree to which a program is understandable without much difficulty.
- Software system independence- degree to which program is independent of nonstandard programming language features, operating system characteristics and other environment constraints.
- Traceability- ability to trace a design representation or actual program component back to initial objectives.
- Training- degree to which the software is user-friendly to new users.

There are various 'checklists' for software quality. One of them was given by Hewlett-Packard that has been given the acronym FURPS – for Functionality, Usability, Reliability, Performance and Supportability. Functionality is measured via the evaluation of the feature set and the program capabilities, the generality of the functions that are derived and the overall security of the system.

Considering human factors, overall aesthetics, consistency and documentation assesses usability. Reliability is figured out by evaluating the frequency and severity of failure, the accuracy of output results, the mean time between failure (MTBF), the ability to recover from failure and the predictability of the program.

Performance is measured by measuring processing speed, response time, resource consumption, throughput and efficiency. Supportability combines the ability to extend the program, adaptability, serviceability or in other terms maintainability and also testability, compatibility, configurability and the ease with which a system can be installed.

1.4 SOFTWARE QUALITY MATRICS

We best manage what we can measure. Measurement enables the Organization to improve the software process; assist in planning, tracking and controlling the software project and assess the quality of the software thus produced. It is the measure of such specific attributes of the process, project and product that are used to compute the software metrics. Metrics are analyzed and they provide a dashboard to the management on the overall health of the process, project and product. Generally, the validation of the metrics is a continuous process spanning multiple projects. The kind of metrics employed generally account for whether the quality requirements have been achieved or are likely to be achieved during the software development process. As a quality assurance process, a metric is needed to be revalidated every time it is used. Two leading firms namely, IBM and Hewlett-Packard have placed a

great deal of importance on software quality. The IBM measures the user satisfaction and software acceptability in eight dimensions which are capability or functionality, usability, performance, reliability, ability to be installed, maintainability, documentation, and availability. For the Software Quality Metrics the Hewlett-Packard normally follows the five Juran quality parameters namely the functionality, the usability, the reliability, the performance and the serviceability. In general, for most software quality assurance systems the common software metrics that are checked for improvement are the Source lines of code, cyclical complexity of the code, Function point analysis, bugs per line of code, code coverage, number of classes and interfaces, cohesion and coupling between the modules etc.

Metrics

There are many forms of metrics in SQA but they can easily be divided into three categories: product evaluation, product quality, and process auditing.

Product Evaluation Metrics – Basically, this type of metric is actually the number of hours the SQA member would spend to evaluate the application. Developers who might have a good application would solicit lesser product evaluation while it could take more when tackling an application that is rigged with errors. The numbers extracted from this metric will give the SQA team a good estimate on the timeframe for the product evaluation.

Product Quality Metrics – These metrics tabulates all the possible errors in the application. These numbers will show how many errors there are and where do they come from. The main purpose of this metric is to show the trend in error. When the trend is identified the common source of error is located. This way, developers can easily take care of the problem compared to answering smaller divisions of the problem. There are also metrics that shows the actual time of correcting the errors of the application. This way, the management team who are not entirely familiar with the application.

Process Audit Metrics – These metrics will show how the application works. These metrics are not looking for errors but performance. One classic example of this type of metric is the actual response time compared to the stress placed on the application. Businesses will always look for this metric since they want to make sure the application will work well even when there are thousands of users of the application at the same time.

There are lots of options on what standard to be used in developing the plan for Software Quality Assurance. But on metrics, the numbers are always constant and will be the gauge whether the application works as planned.

Common software metrics include:

- Bugs per line of code
- Code coverage
- Cohesion
- Coupling
- Cyclomatic complexity
- Function point analysis
- Number of classes and interfaces
- Number of lines of customer requirements
- Order of growth
- Source lines of code
- Robert Cecil Martin's software package metrics

Software Quality Metrics focus on the process, project and product. By analyzing the metrics the organization the organization can take corrective action to fix those areas in the process, project or product which are the cause of the software defects.

The de-facto definition of software quality consists of the two major attributes based on intrinsic product quality and the user acceptability. The software quality metric encapsulates the above two attributes, addressing the mean time to failure and defect density within the software components. Finally it assesses user requirements and acceptability of the software. The intrinsic quality of a software product is generally measured by the number of functional defects in the software, often referred to as bugs, or by testing the software in run time mode for inherent vulnerability to determine the software "crash" scenarios. In operational terms, the two metrics are often described by terms namely the defect density (rate) and mean time to failure (MTTF).

Although there are many measures of software quality, correctness, maintainability, integrity and usability provide useful insight.

Correctness A program must operate correctly. Correctness is the degree to which the software performs the required functions accurately. One of the most common measures is Defects per KLOC. KLOC means thousands (Kilo) Of Lines of Code.) KLOC is a way of measuring the size of a computer program by counting the number of lines of source code a program has.

Maintainability : Maintainability is the ease with which a program can be correct if an error occurs. Since there is no direct way of measuring this an indirect way has been used to measure this. MTTC (Mean time to change) is one such measure. It measures when a error is found, how much time it takes to analyze the change, design the modification, implement it and test it.

Integrity : This measure the system's ability to with stand attacks to its security. In order to measure integrity two additional parameters are threat and security need to be defined. Threat – probability that an attack of certain type will happen over a period of time. Security – probability that an attack of certain type will be removed over a period of time. Integrity = Summation [(1 - threat) X (1 - security)]

Usability : How usable is your software application ? This important characteristic of your application is measured in terms of the following characteristics:

- Physical / Intellectual skill required to learn the system
- time required to become moderately efficient in the system.
- the net increase in productivity by use of the new system.
- subjective assessment(usually in the form of questionnaire on the new system).

Standard for the Software Evaluation

In context of the Software Quality Metrics, one of the popular standards that addresses the quality model, external metrics, internal metrics and the quality in use metrics for the software development process is ISO 9126.

Defect Removal Efficiency

Defect Removal Efficiency (DRE) is a measure of the efficacy of your SQA activities.. For eg. If the DRE is low during analysis and design, it means you should spend time improving the way you conduct formal technical reviews.

$$DRE = E / (E + D)$$

Where E = No. of Errors found before delivery of the software and D = No. of Errors found after delivery of the software.

Ideal value of DRE should be 1 which means no defects found. If you score low on DRE it means to say you need to re-look at your existing process. In essence DRE is a indicator of the filtering ability of quality control and quality assurance activity. It encourages the team to find as many defects before they are passed to the next activity stage. Some of the Metrics are listed out here:

Test Coverage = Number of units (KLOC/FP) tested / total size of the system Number of tests per unit size = Number of test cases per KLOC/FP Defects per size = Defects detected / system size Cost to locate defect = Cost of testing / the number of defects located Defects detected in testing = Defects detected in testing / total system defects Defects detected in production = Defects detected in production/system size Quality of Testing = No. of defects found during Testing/(No. of defects found during testing + No of acceptance defects found after

delivery) *100 System complaints = Number of third party complaints / number of transactions processed Effort Productivity = Test Planning Productivity = No of Test cases designed / Actual Effort for Design and Documentation Test Execution Productivity = No of Test cycles executed / Actual Effort for testing Test efficiency= (number of tests required / the number of system errors).

1.2 & 1.3 Check your progress

I. Fill in the Blanks

1. Plan, Do, Check, and Act are steps to determine _____ .
2. Quality Factor _____ is effort required to test the programs for their functionality.
3. _____ is degree to which one can extend architectural, data and procedural design.
4. Two leading firms namely, _____ and _____have placed a great deal of importance on software quality.
5. Software Quality _____ focus on the process, project and product.

1.5 SOFTWARE PROCESS IMPROVEMENT

Process Areas in Capability Maturity Model (CMM)

The Capability Maturity Model Integration (CMMI), based process improvement can result in better project performance and higher quality products. A Process Area is a cluster of related practices in an area that, when implemented collectively, satisfy a set of goals considered important for making significant improvement in that area.

In CMMI, Process Areas (PAs) can be grouped into the following four categories to understand their interactions and links with one another regardless of their defined level:

Process Management : It contains the cross-project activities related to defining, planning, resourcing, deploying, implementing, monitoring, controlling, appraising, measuring, and improving processes. Process areas are :

- Organisational Process Focus.
- Organisational Process Definition.
- Organisational Training.
- Organisational Process Performance.
- Organisational Innovation and Deployment.

Project Management : The process areas cover the project management activities related to planning, monitoring, and controlling the project. Process areas are :

- ✓ Project Planning.
- ✓ Project Monitoring and Control.
- ✓ Supplier Agreement Management.
- ✓ Integrated Project Management for IPPD (or Integrated Project Management).
- ✓ Risk Management.
- ✓ Integrated Teaming.
- ✓ Integrated Supplier Management.
- ✓ Quantitative Project Management.

Engineering : Engineering process areas cover the development and maintenance activities that are shared across engineering disciplines. Process areas are :

- ✓ Requirements Development.
- ✓ Requirements Management.
- ✓ Technical Solution.
- ✓ Product Integration.
- ✓ Verification.
- ✓ Validation.

Support : Support process areas cover the activities that support product development and maintenance. Process areas are :

- ✓ Process and Product Quality Assurance.
- ✓ Configuration Management.
- ✓ Measurement and Analysis.
- ✓ Organisational Environment for Integration.
- ✓ Decision Analysis and Resolution.
- ✓ Causal Analysis and Resolution.

1.6 PROCESS AND PRODUCT QUALITY ASSURANCE (PPQA) PROCESS AREA IN CMMI

The purpose of Process and Product Quality Assurance (PPQA) is to provide staff and management with objective insight into processes and associated work products. A Support Process Area at Maturity Level 2.

The Process and Product Quality Assurance process area involves the **following activities:**

- ✓ Objectively evaluating performed processes, work products, and services against applicable process descriptions, standards, and procedures.
- ✓ Identifying and documenting noncompliance issues.
- ✓ Providing feedback to project staff and managers on the results of quality assurance activities.
- ✓ Ensuring that noncompliance issues are addressed.

The Process and Product Quality Assurance process area supports the delivery of high-quality products and services by providing project staff and managers at all levels with appropriate visibility into, and feedback on, processes and associated work products throughout the life of the project.

Specific Goals and Practices

SG 1 Objectively Evaluate Processes and Work Products

Adherence of the performed process and associated work products and services to applicable process descriptions, standards, and procedures is objectively evaluated.

SP 1.1 Objectively Evaluate Processes.

Objectively evaluate the designated performed processes against the applicable process descriptions, standards, and procedures. Objectivity in quality assurance evaluations is critical to the success of the project. A description of the quality assurance reporting chain and how it ensures objectivity should be defined.

SP 1.2 Objectively Evaluate Work Products and Services.

Objectively evaluate the designated work products and services against the applicable process descriptions, standards, and procedures. The intent of this subpractice is to provide criteria, based on business needs,

such as the following:

- What will be evaluated during the evaluation of a work product?
- When or how often a work product will be evaluated?
- How the evaluation will be conducted?
- Who must be involved in the evaluation?

SG 2 Provide Objective Insight

Noncompliance issues are objectively tracked and communicated, and resolution is ensured.

- SP 2.1 Communicate and Ensure Resolution of Noncompliance Issues.

Communicate quality issues and ensure resolution of noncompliance issues with the staff and managers. Noncompliance issues are problems identified in evaluations that reflect a lack of adherence to applicable standards, process descriptions, or procedures. The status of noncompliance issues provides an indication of quality trends. Quality issues include noncompliance issues and results of trend analysis.

When local resolution of noncompliance issues cannot be obtained, use established escalation mechanisms to ensure that the appropriate level of management can resolve the issue. Track noncompliance issues to resolution.

- SP 2.2 Establish Records.

Establish and maintain records of the quality assurance activities. Typical Work Products are evaluation logs, quality assurance reports, status reports of corrective actions and reports of quality trends.

1.7 THE SEI PROCESS CAPABILITY MATURITY MODEL, ISO, SIX-SIGMA

SEI = 'Software Engineering Institute' at Carnegie-Mellon University; initiated by the U.S. Defense Department to help improve software development processes.

CMM = 'Capability Maturity Model', now called the CMMI ('Capability Maturity Model Integration'), developed by the SEI. It's a model of 5 levels of process 'maturity' that determine effectiveness in delivering

quality software. It is geared to large organizations such as large U.S. Defense Department contractors. However, many of the QA processes involved are appropriate to any organization, and if reasonably applied can be helpful. Organizations can receive CMMI ratings by undergoing assessments by qualified auditors.

Level 1 - characterized by chaos, periodic panics, and heroic efforts required by individuals to successfully complete projects. Few if any processes in place; successes may not be repeatable.

Level 2 - software project tracking, requirements management, realistic planning, and configuration management processes are in place; successful practices can be repeated.

Level 3 - standard software development and maintenance processes are integrated throughout an organization; a Software Engineering Process Group is is in place to oversee software processes, and training programs are used to ensure understanding and compliance.

Level 4 - metrics are used to track productivity, processes, and products. Project performance is predictable, and quality is consistently high.

Level 5 - the focus is on continouous process improvement. The impact of new processes and technologies can be predicted and effectively implemented when required.

Perspective on CMM ratings: During 1997-2001, 1018 organizations were assessed. Of those, 27% were rated at Level 1, 39% at 2, 23% at 3, 6% at 4, and 5% at 5. (For ratings during the period 1992-96, 62% were at Level 1, 23% at 2, 13% at 3, 2% at 4, and 0.4% at 5.) The median size of organizations was 100 software engineering / maintenance personnel; 32% of organizations were U.S. federal contractors or agencies. For those rated at Level 1, the most problematical key process area was in Software Quality Assurance.

ISO = 'International Organisation for Standardization' - The ISO 9001:2008 standard (which provides some clarifications of the previous standard 9001:2000) concerns quality systems that are assessed by outside auditors, and it applies to many kinds of production and manufacturing organizations, not just software. It covers documentation, design, development, production, testing, installation, servicing, and other processes. The full set of standards consists of: (a)Q9001-2008 - Quality Management Systems: Requirements; (b)Q9000-2005 - Quality Management Systems: Fundamentals and Vocabulary; (c)Q9004-2009 -

Quality Management Systems: Guidelines for Performance Improvements. To be ISO 9001 certified, a third-party auditor assesses an organization, and certification is typically good for about 3 years, after which a complete reassessment is required. Note that ISO certification does not necessarily indicate quality products - it indicates only that documented processes are followed.

ISO 9126 is a standard for the evaluation of software quality and defines six high level quality characteristics that can be used in software evaluation. It includes functionality, reliability, usability, efficiency, maintainability, and portability.

IEEE = 'Institute of Electrical and Electronics Engineers' - among other things, creates standards such as 'IEEE Standard for Software Test Documentation' (IEEE/ANSI Standard 829), 'IEEE Standard of Software Unit Testing (IEEE/ANSI Standard 1008), 'IEEE Standard for Software Quality Assurance Plans' (IEEE/ANSI Standard 730), and others.

ANSI = 'American National Standards Institute', the primary industrial standards body in the U.S.; publishes some software-related standards in conjunction with the IEEE and ASQ (American Society for Quality).

Six Sigma is a methodology of quality management that gives a company tools for business processes improvement. This approach allows to manage quality assurance and business processes more effectively, and reduce costs and increase company profits. *The fundamental principle* of Six Sigma approach is the customer satisfaction through implementing defects-free business processes and products (3.4 or fewer defective parts per million).

Six Sigma approach determines factors that are important for product and service quality. This approach contributes to reduction of the business process deviation, improvement of opportunities and increase of production stability.

There are five stages in Six Sigma Project:

1. Defining

The first stage of Six Sigma project is to define the problem and deadlines to solve this problem. The team of specialists considers a business process (e.g., production process) in details and identifies defects that should be erased. Then the team generates a list of tasks to improve the business process, project boundaries, customers, their product and service requirements and expectations.

2. Measuring

On the second stage the business process is to be measured and current performance is to be defined. The team collects all data and compares it to customer requirements and expectations. Then the team prepares measures for future large-scale analysis.

3. **Analyzing**

As soon as the data is put together and the whole process is documented, the team starts analysis of the business process. The data collected on stage two “Measuring” are determine root reasons of defects/problems and identify gaps between current performance and new goal performance. Usually the team specialists begin with defining the fields in which employees make mistakes and cannot take effective control of the process.

4. **Improving**

On this stage the team analyzes the business process and works up some recommendations, solutions and improvements to erase defects/problem or achieve desired performance level.

5. **Controlling**

On the final stage of Six Sigma Project the team creates means of control of the business process. It allows the company to hold and extend scale of transformations.

Other software development/IT management process assessment methods besides CMMI and ISO 9000 include SPICE, Trillium, TickIT, Bootstrap, ITIL, MOF, and CobiT.

1.8 PROCESS CLASSIFICATION

The processes can be classified to structure service level agreements. The process classes shown in table have been identified so far in past and current projects. Depending on the concrete service there may be some of the classes missing but the complexes the service the more of them are necessary.

Services must satisfy some quality criteria to reach a service level acceptable to the customer. For example, in the presented scenario the response time must be small enough to query information during the sales conversation. Management processes must reach reasonable service levels, too, e.g. the problem resolution process must fix an error in a certain amount of time.

Table: Process Classes

Class	*Description*
provisioning	installation of the service
usage	normal usage of the service
operation	service management of the provider

maintenance	plan able activities needed for service sustainment
accounting	collection and processing of accounting data
change management	extension, cancellation and change of service elements
problem management	notification, identification and resolution of service malfunction
SLA management	management of SLA contents
customer care	service reviews and user support
termination	De-installation of the service

Before the service can be used it must be installed. The *provisioning* class includes installation, test, acceptance and if necessary migration processes. Timeliness is an important quality parameter for processes in this class. *Usage* is the most common interaction. Service levels define for example the response time of the service.

Operation processes are invisible to the customer most of the time, but quality parameters are needed for them anyway, for example, the amount of time to detect a malfunction of a POP for ISDN calls is a quality criteria of service operation. As the service is usually at least limited during *maintenance* activities the customer wants to limit e.g. the time slot for maintenance. The method used for *accounting* purposes is an important fact because it influences the price of the service.

Minor changes of the service are not uncommon in long-term relationships. This could be the extension of resources or addition of further dealers in the presented scenario. Thus a *change management* is necessary. A well working *problem management* is a very important factor in outsourcing. Problems can be solved faster if both partners are working together smoothly using well defined processes.

Often the requirements of the customer change during lifetime of the SLA. If these changes are known but not the point in time it is useful to specify a *SLA management* with processes for announcing the need, negotiating and changing the SLA. Communication on strengths and weaknesses of the service, its quality or future optimizations should be some of the subjects of regular *customer care* contacts. Besides, a help desk or other support offers are usually necessary for complex services.

Termination of processes for a relationship usually need to be specified if equipment of the provider is installed at customer locations or if support of the provider is needed for migration to the following service.

Additionally, it is useful to define some basic processes which are needed in many classes. Such processes are for example documentation, feedback and escalation mechanisms.

1.4 - 1.8 Check Your Progress

Answer the following in one sentence

1. Name the Process areas categories of CMMI.

   .....

   .....

2. The Process and Project Quality works on.

   .....

   .....

1.9 SUMMARY

Quality is the activities designed to improve the organization and its services. Quality is essentially about learning what you are doing well and doing it better. Quality assurance is the process of verifying or determining whether products or services meet or exceed customer expectations. Quality assurance is a process-driven approach with specific steps to help define and attain goals. This process considers design, development, production, and service. Quality control is the observation techniques and activities used to fulfill requirements for quality. The various factors, which influence the software, are termed as software factors. They can be broadly divided into two categories. The classification is done on the basis of measurability. The first category of the factors is of those that can be measured directly such as number of logical errors and the second category clubs those factors which can be measured only indirectly for example maintainability but the each of the factors are to be measured to check for the content and the quality control. Two leading firms namely, IBM and Hewlett-Packard have placed a great deal of importance on software quality. Software Quality Metrics focus on the process, project and product. In operational terms, the two metrics are often described by terms namely the defect density (rate) and mean time to failure (MTTF).Although there are many measures of software quality, correctness, maintainability, integrity and usability provide useful insight. Defect Removal Efficiency (DRE) is a measure of the efficacy of your SQA activities.DRE is a indicator of the filtering ability

of quality control and quality assurance activity. Process Areas in Capability Maturity Model (CMM). The Capability Maturity Model Integration (CMMI), based process improvement can result in better project performance and higher quality products. SEI = 'Software Engineering Institute', ISO = 'International Organization for Standardization', IEEE = 'Institute of Electrical and Electronics Engineers', ANSI = 'American National Standards Institute' and Six Sigma are few standards for Software Engineering and Software Quality Assurance.

1.10 CHECK YOUR PROGRESS- ANSWERS

1.2 & 1.3

1. Quality Assurance
2. Testability
3. Expandability
4. IBM and Hewlett-Packard
5. Metrics

1.4 - 1.8

Process Management, Project Management, Engineering and support

1. Goals and practices.

1.11 QUESTIONS FOR SELF – STUDY

Q1. Explain PPQA in CMMI.

Q2. Write the SEI Process Capability Maturity model.

Q3. Explain the five stages in Six Sigma.

❁❁❁

NOTES

Chapter 2

Software Quality Assurance

2.0 OBJECTIVES

Dear students, after having overview of software engineering and concepts of software quality assurance, this chapter focuses on need of SQA. After studying this chapter you will be able to-

- Describe SQA Activities
- Explain the building blocks of SQA
- Discuss the SQA Planning & Standards

2.1 INTRODUCTION

SQA should be utilized to the full extend with traceability of errors, cost effiecient . There are the principles behind a highly efficient SQA procedure. Every business should use SQA and use it to the best as they can. They can easily guide the application to be the best procedure. Everything should be “spick and span” when SQA has been executed. Since it is internationally recognized, the application should not only have a local impact but global acceptance at the same time. Businesses that develop and application and setting it for global standard will have a high reputation and the salability is even better. SQA definitely has benefits that will launch the company towards the global standard it needs.

2.2 NEED OF SQA

There are so many reasons why a company should consider SQA. It is all about business survival and SQA is just one of the many tools the company should effectively use. And just like a tool, it has to be effectively used to its maximum. If the tool is not used to its full extent, the tool will just be a financial burden to the company. SQA should also be utilized at the same way – to its full extent.

One of the reasons why businesses and companies opt out in using SQA is their inability to realize who effective an SQA when properly used. The following are the reasons why SQA should be used by any company before releasing their application to their intended users:

Traceability of Errors

Without using SQA, developers can still locate the errors of the application. Since their familiarity with the application is impeccable, they will have the ability to find a remedy to the problem. SQA on the other hand does not just look for answers to their problems.

SQA looks for the reason why the error occurred. Looking for errors is really easy but tracing the roots of the problem is another thing. The SQA team should be able to tell which practice has started the problem. Most of the time, the errors is not rooted on the execution of the coding but it could be found in the philosophy in developing the application.

Cost Efficient

This idea is saving money is very debatable in SQA. Using SQA means you will have to hire another set of people to work with you. Most of the SQA today are 3rd party companies that are only there to work on the specific project. Manpower will always cost money. Manpower does not even cover the cost of the tools the SQA team might use. Unfortunately most of the SQA tools are expensive especially the most efficient tools.

On the other hand, consider the idea of locating the bugs and errors in the application before they are released. Every application that are used in a business setting or sold to the consumers should have an ideal of 0% error. Even though this is rarely attained, SQA should be able to reduce errors to less than 1%.

Considering the effect on productivity together with customer satisfaction, you can always tell that preventing the problem is always better compared to answering them. SQA could practically save your business and boost it to success. Without them, there is a higher possibility that the application will have errors which will discourage your customers from purchasing the application. In a business setting, an erroneous application will lead to slower productivity since instead of

helping their customers; they might end up troubleshooting their application most of the time.

Flexible Solutions

SQA could easily provide solutions to the problem since they look for the root of the problem instead of just answering them. By providing the root of the problem, they have the ability to provide solutions to these problems fast. But instead of one solution only, the SQA team should be able to provide more than one solution.

Finding the root of the problem means they can easily provide more than one solution. Most of the SQA companies offer web-based solutions to their concerns and aside from that, on-site solutions are also provided by most companies to ensure that the company could have as much options for solutions.

Better Customer Service

One of the ultimate goals of any businesses is to provide the best customer service possible. SQA could help these companies realize that goal. Most of the transactions in today's business world today are IT related so an efficient application should be employed in the first place. In order to do that, the application should go under rigorous test. Testers could do that but again, that is only looking for the answer of the problem.

SQA goes more than simple solution to the problem. The SQA team takes note at every step of the application development process. Through this, they will know what are the practices that might go out against the industry standards.

Aside from monitoring, SQA also has a good reputation in looking after the development of the application. There are universal standards that guide every SQA team on how to monitor and guide the application for its success. Every step of the way has been standardized.

Even the recommended documents are carefully planned out to ensure that that feed back is working as planned. The end result is a globally recognized product developed with certain standards. The SQA methodologies are ISO certified which means, the practice of monitoring the development of a software or application is recognized by some of the world's biggest companies.

Innovation and Creativity

Since the SQA team is here to standardize how their work is done, it still fosters innovation and creativity for developing the product. Everyone is given a free hand in developing the application and the SQA team is there to help them standardize the ideas.

Most of the time, developers would feel constricted in building applications because of the metrics proposed by the developers. On the

other hand, developers should have the ability to be more creative as long as the SQA team is there. The SQA team on the other hand has the ability to foster drive as everything will be under the guidance of the team.

Software Principles

The SQA team also has to follow certain principles. As a provider of quality assurance for procedures and application, they need to have a strong foundation on what to believe in and what to stand for. These principles will ensure that the application will live up to the expectations of the client and the users. Like the application, the SQA principles run on the background but eventually dictate how the system will be tackled. The following are some of the most powerful principles that can be used for proper execution of software quality assurance:

Feedback – In gist, the faster the feedback the faster the application will move forward. An SQA principle that uses rapid feedback is assured of success. Time will always be the best friend and the most notorious enemy of any developer and it's up to the SQA team to give the feedback as soon as possible. If they have the ability to get the feedback of their applications as soon as possible, then the chance of developing a better application faster is possible.

Focus on Critical Factor – This principle has so many meanings; first it just means that some of the factors of the software being developed are not as critical compared to other. That means SQA should be focused on the more important matters.

Secondly, SQA's measurement should never be universal in the sense that every factor in the application should not have the same treatment. One great example of this is the treatment of the specific functions compared to the skin or color of the interface. Clearly, the function should have more focus compared to a simple skin color.

Multiple Objectives – This is partly a challenge as well as risk for the SQA team. At the start of the SQA planning, the team should have more than one objective. If you think about it, it could be very dangerous however it is already a common practice. But what is emphasized here is that each objective should be focused on. As much as possible a matrix should be built by the SQA so that it could track the actual actions that relates to the objective.

Evolution – Reaching the objective is really easy but every time something new happens, it should be always noted. Evolution is setting the benchmark in each development. Since the SQA team is able to mark every time something new is done, evolution is monitored.

The good thing about this principle is for future use. Whenever a benchmark is not reached, the SQA team should be able to study their previous projects. Evolution should be able to inform and educate the SQA team while working on the project.

Quality Control – By the name itself, Quality Control is the pillar for Software Quality Assurance. Everything needs to have quality control – from the start to the finish. With this principle there has to be an emphases on where to start. The biggest and the tightest quality control should be executed as early as possible.

For example, when the SQA team receives the SR (software requirements document) the intensity of quality control should be at the start. Of course quality control will still be executed until the end but developers should take into account that anything that starts out real bad could never take off. It's better to know what's wrong at first than to find that out later.

Motivation – There is not substitute than to have the right people who has the will to do their job at all times. When they have the right mindset the willingness to do it, everything will just go through. Work will definitely be lighter, expertise will be seen and creativity is almost assured when everyone has the drive and passion in their line of work. Quality assurance is a very tedious task and will get the most out of the person if they are not dedicated to their line of work.

Process Improvement – Every project of the SQA team should be a learning experience. Of course each project will give us the chance to increase our experience of SQA but there's more to that. Process improvement fosters the development of the actual treatment of the project. Every project has a unique situation that will give the SQA team a chance to experience something new. This "new" will never be translated to something good if they are not documented for future references. Learning should not only be based on individual experience but also on company's ability to adapt to the new situation and use it for future references.

Persistence – There is no perfect application. The bigger they get, the more error there could be. The SQA team should be very tenacious in looking for concerns in every aspect of the software development process. Even with all the obstacles everyone would just have to live with the fact that every part should be scrutinized without hesitation.

Different Effects of SQA – SQA should go beyond software development. A regular SQA will just report for work, look for errors and leave. The SQA team should be role models in business protocols at all times. This way, the SQA does not only foster perfection in the application but also in their way of life. That seemed to be quite off topic but believe me; when people dress and move to success, their work will definitely reflect with it.

Result-focused – SQA should not only look at the process but ultimately its effect to the clients and users. The SQA process should always look for results whenever a phase is set.

These are the principles that every SQA plan and team should foster. These principles tell encourages dedication towards work and patience not necessarily for perfection but for maximum efficiency.

2.3 SQA ACTIVITIES

SQA is composed of a variety of tasks linked with 2 dissimilar constituencies: SQA group that has responsibility for quality assurance planning, record keeping, oversight, analysis and reporting and the other are software engineers, who do technical work. The character of a SQA group is to help the software team in attaining a high-class product.

The SQA group prepares a SQA plan that identifies:

- Assessments to be carried out,
- Reviews and audits to be executed,
- Standards those are relevant to the project,
- Measures for error reporting and tracking,
- Credentials to be produced by the SQL group, and
- Amount of feedback provided to the software project team.

2.2 & 2.3 Check Your Progress

I. State the following statements are True or False

1. SQA should be able to reduce to the errors to less than 5%.

2. The SQA team takes note at every step of the application development process.

3. Software Engineers are not involved in SQA.

4. The main aim of SQA is to remove errors and not to reach the perfection.

5. Quality control is pillar for Software Quality assurance.

2.4 BUILDING BLOCKS OF SQA

2.4.1 SQA Project Metrics

The application is only as good as its numbers. It is a harsh reality but everything has to come down to the numbers. Although we can enumerate tens or hundreds of features in a single application, it will all be for nothing if the metrics do not live up according to expectation. The SQA team should ensure that the expected metrics will be posted. At the same time, the SQA team should also select the right tool to gauge the application.

There are hundreds of applications that can measure the application but it is quite rare to find the right application to provide the right information.

To fully gauge the software's ability, a number of metrics should be attained. A regular testing application will just show errors of the software as well its ability to withstand constant and heavy traffic. However when you take a closer look at the application, you will realize these are not the only numbers that you need to know if the application actually works. The SQA team should know the numbers that needs to be shown to the developers and to the management.

Metrics

In the previous chapter, the metrics provided were referred to the SQA's ability to influence the application. This time, the metrics will only refer to the application alone. To gauge the actual application, the metrics are divided into four categories. Each category has attributes with specific metrics ensuring that the attribute is achieved.

Quality of Requirements – These set of metrics will show how well the application is planned. Among the requirements of quality first and foremost is completeness. Remember that requirements are geared to answer all the problems and concerns of the users. Therefore, its main aim is to answer all the concerns.

One of metrics found in this category is the completeness of the requirements. As this is in the planning stage, the ability to be understood is also important. The SQA team should gauge if the document are well written and useful.

Lastly, the requirements that were written by the developers and the intended users should be traceable. No one can just place a requirement out of whim since each of the requirements should have a logical root or explanation.

Product Quality – These set of metrics will gauge the ability of the developers to formulate codes and functions. The SQA team will first gauge how simple the application has been written. The software has to be written in a simple manner. It might sound contradicting that even a highly complex application could come in as simple.

What the SQA team is looking for from the application is the logical formulation of the codes. When the application is logically coded, it has two repercussions which are also gauged by SQA.

First is the maintainability. If the application could be easily understood, troubleshooting is very easy. Reusability of the application is also emphasized. Since the codes could easily be distinguished, developers can use parts of the application to be implemented to another program.

2.4.2 SQA Software and Tools

In quality assurance, it is always important to get all the help we could get. In other industries, developers could easily check the products manually and discard those that do not meet the standard. The length and the width of the product are checked to maintain standardization of the product. Others use special machines to check the product. With tools and machines, they can easily set a standard with their products.

That also goes the same with software and applications. Although it does not use physical machines, applications go through rigorous testing before they are released to the public even for beta testing. The tools used in SQA are generally testing tools wherein an application is run through a series of tests to gauge the performance of the application.

The tools used in SQA vary in purpose and performance. These applications range from testing the code or running the application under great stress. These tools are employed to test the application and produce numbers and statistics regarding the actual application. Through these numbers, the SQA team and their developers will know if the application has lived up according to the targeted performance.

Like most developers each SQA team has their preferred tools for application testing. Based on their belief and expertise, the SQA team will usually give the owners or business managers a free hand on what type of testing tool to use.

Notable SQA Tools

The following are some of the renowned SQA tools and applications. There are still hundreds out there but the following tools have been around for years and have been used by thousands or probably millions of testers.

WinRunner – Developed by HP, WinRunner is a user friendly application that can test the applications reaction from the user. But other than measuring the response time, WinRunner can also replay and verify every transaction and interaction the application had with the user. The application works like a simple user and captures and records every response the application does.

LoadRunner – Developed by HP, LoadRunner is one of the simple applications that can test the actual performance of the application. If you are looking for a program to test your application's tolerance to stress, LoadRunner is your tool. It has the ability to work like thousands of users **at the same time –** testing the stress of the application.

QuickTest Professional – If you have worked with WinRunner you surely have bumped in with this tool. Built by HP, QuickTest emulates the actions of users and exploits the application depending on the procedure set by testers. It can be used in GUI and non-GUI websites and

applications. The testing tool could be customized through different plug-ins.

Mercury Test Director – An all-in-one package, this web-based interface could be used from start to end in testing an application or a website. Every defect will be managed according to their effect to the application. Users will also have the option to use this exclusively for their application or use it together with wide array of testers.

Silktest – Although available in limited operating system, Silktest is a very smart testing tool. Silktest lists all the possible functions and tries to identify the function one by one. It can be implemented in smaller iterations as it translate the available codes into actual objects.

Bugzilla – Developed by Mozilla, this open source testing tool works as the name suggests. Bugzilla specializes in detecting bugs found in the application or website. Since the application is open-source it can be used freely and it is availability in different OS makes it even a viable alternative for error tracking. The only downside is it has a long list of requirements before it could run.

Application Center Test – Also known as ACT, this testing tool was developed by Microsoft using ASP.NET. This application is primarily used for determining the capacity of the servers that handle the application. Testers can test the server by asking constant requests. A customized script either from VB or JS could be used to test the server's capacity.

OpenSTA – Another open source tool, testers can easily launch the application and use it for testing the application's stress capacity. The testing process could be recorded and testing times could be scheduled. Great for websites that needs daily maintenance.

QARun – Instead of an application, QARun is actually a platform where you can build your own testing application. QARun could be easily integrated with the application so that it could easily sync with the releases and checks the application or website every time something new is introduced.

The documentation that supports the application is also gauged and the comment within the coding is also gauged. It is not important if there lots of comments but what is important is that the comments are clear and could be found in every function.

Implementation Capability –

The general coding behavior of the developers is also gauged by the SQA team. The metrics used in this classification is based on the SDLC used by the developers. The SQA team will rate the developer's ability to finish each action in the stage of the SDLC on time.

The staff hours are rated according to the need of the stage. Each phase of the SDLC will have their time requirement – some will just need

a day or two while others will require a month to finish. The time required in every stage is set by the project manager.

Aside from the time required for completion, the developers are also rated if they ever finish a task at all. There are tasks that are necessary for the application to run while there are tasks that could be offset by another task. Although that's a convenience, it will actually trigger a domino effect so the whole operation could be placed in jeopardy.

Software Efficiency – Last but maybe the most important is to ensure that the application do not have any errors at all. This is basically a major pillar to any application. Without errors, the software will never give any hackers a chance to infiltrate the system. Usually, the SQA team will develop test cases wherein the error is highlighted. The developers are then rated how fast they could locate the root of the problem and how fast they could built a fix to the problem.

These are the metrics that every software developer will have to go through. The better the rating, the better the application would work.

2.4.3 Selecting the Right Quality Model

There are so many quality models to choose from. Each of them has their own advantage and disadvantages. It is up to the SQA team to select which application that will give the clients a good idea how the application works.

In order to select a good Quality Model is no trick actually. Every application developed has a "core" or a feature that differentiates the application from other software.

The SQA team should just focus on that feature and look for the quality model that can fully gauge the feature. It is a very simple idea but will do wonders especially when the SQA team is trying to prove the validity of the application against the feature needed by the clients.

The only disadvantage of this technique is that the developers might be focusing too much in the feature. However, all of the Quality Models could easily gauge every aspect of the application. That means every SQA team can easily gauge the software without any problem. Finding errors in the application is a great challenge to the SQA team. But with the right tools the software could be easily gauged and the application will work as expected.

2.4.4 Selecting Right SQA Tool

The tools that are listed above are only a very small chunk of the hundreds of testing tools available as a free or licensed application. Among the dilemmas of the SQA team is to select the proper testing tool for the particular website or application.

In selecting the right tool, testers and SQA managers should always start with their objective. Although usually the testers' objective is to know every bug and error related to the application, testers and developers should also consider writing benchmarks. Through this benchmark, they can assure the application works as expected. Every tester should understand that each application might require a completely different testing tool. Experience will always tell the testers that a simple automation tool will never work in complex applications.

Aside from this consideration, testers also have the choice to use more than one testing tool. But before getting all the possible testing tools, the SQA team should consider what metrics they are looking for. By determining the numbers they need and the benchmark they are looking for they can easily select the testing tools they can use.

The success of SQA can be determined based on the applications they use. In fact, it can be safely said that the SQA team is only as good as the software they use. Although they can check the application manually, it will take time and it will always be prone to human error. With the right testing tool, the application is tested faster with better accuracy.

2.4.5 SQA Planning and Requirements

The scope of Software Quality Assurance or SQA starts from the planning of the application until it is being distributed for the actual operations. To successfully monitor the application build up process, the SQA team also has their written plan. In a regular SQA plan, the team will have enumerated all the possible functions, tools and metrics that will be expected from the application. SQA planning will be the basis of everything once the actual SQA starts.

Without SQA planning, the team will never know what the scope of their function is. Through planning, the client's expectations are detailed and from that point, the SQA team will know how to build metrics and the development team could start working on the application.

When the SQA team starts working, one of the first things they will have to tackle is to determine the developer team's initial output. The intended users or at least part of them will be forwarding information which tells of their expectations of the new software called UR (User Requirements), this document will be the bases.

What is left is to determine the software requirements in the intended application. In business or in any intended application, software requirements are very important since these requirements will tell how an application would work. This will also be the ground work for developers. From these requirements they will know what language, SDLC program and other protocols that will be used to build the applications.

While the developers work on determining the requirements, the SQA team will be monitoring their work to ensure the requirements are according to the expectations.

2.4.6 Technical Activities

One of the responsibilities of the SQA team from the developers is the required user requirements document. The UR document is produced by the intended users who are tapped for this project. This document will inform the developers what the users are expecting from the application. The SQA team will ensure that the document exists before they actually work on the software requirements. The UR document will be the bases for the SR document so it will never make sense if the SR document is produced without consulting the user requirements document first. The SQA team should also ensure that the UR document should be of quality. To develop an efficient software requirement document, the developers should use a known method to determine this type requirement for software development. There are lots of software requirements analysis methods that could be used but developers do not just pick one they prefer. The SQA team should ensure that the analysis method should be reputable or recognized. As much as possible the analysis method should have been published.

Aside from a published analysis method for software requirements, it is also ideal that the analysis method will be supported by different Computer-Aided Software Engineering (CASE) tools. CASE tools are used to develop models for software development and software requirements could easily be determined by with this application. Although it is not required for financial and time frame reasons, CASE tools should be highly considered by developers. The intended metrics for performance is also detailed in software requirements. Above all, metrics should be emphasized since it will tell how fast or how efficient the application should perform. Based on this metrics, the SQA team should be able to determine their testing tools. These documents should reflect the principles and the metrics of the clients and the intended users are looking for.

2.4.7 SQA Management Plans

In the planning and requirements phase, there will be four plans that will be created. These plans will be used in the next stage of software development which is the architectural phase. The Software Quality Assurance Plan (SQAP) for architectural design (AD) is basically the following list of activities to ensure that the preparation of the architectural plan is a success. If any tool or application will be used, the SQA team should point this out in this phase.

- Software Project Management Plans (SPMP) for Architecture Design.

The SQA team should ensure that this management plan will have the specific budget for developing the software. Aside from being specific, the SQA team should also ensure that the estimate should be obtained using scientific methods.

- Software Configuration Management Plans (SCMP) for Architectural Design.

  SCMP is the plan on how the software will be eventually configured. In this phase, the SQA team should ensure that the configuration should have been established at the start.

- Software Verification Management Plan (SVMP) for Architectural Design.

  Like most of the management plans, this one has already been established. But extra information should be sought after by the SQA team. Since this document will be used in the architectural design phase more detailed information is needed. A test plan should be made to ensure that the Architectural Design phase is a success. Also, the SQA team should ensure a general testing system should be established.

Output and Principles

At the end of the planning phase, developers should be able to produce a document that has the details needed for the Architectural Design phase. The SQA team should check if the document has the following characteristics:

- **Consistency –** This is especially applicable if the output does not use any CASE tools. Case tools should easily figure out inconsistencies and it is up to the SQA team to determine this consistency if no CASE tools was used in this phase.
- **Verifiable –** Each specific requirement in this phase should be tested either by a popular tool or verifiable facts from the past.
- **Complete –** The SR or software requirements is the end of the requirement phase. To declare that this phase is complete, the SQA team should check with the UR (user requirements) and see to it that all the things needed by the user are answered. If possible, a matrix should be created to track each answer of the SR to the UR built by the users.

In the end, the SQA team, the developers and representatives of the users will come together for a formal review of the documents. By sitting together, everyone will know if the things needed in the software are covered or will come together with the application.

2.5 SQA PLANNING AND STANDARDS

Planning is one of the most important aspects of Software Quality Assurance. The entire operation of the SQA team depends on how well their planning is done. In smaller businesses, planning might not really dictate the flow of SQA but in larger businesses, SQA Planning takes on center stage. Without it, each component or department that works on the application will be affected and will never function.

In gist, SQA Planning tackles almost every aspect of SQA's operation. Through planning, each member and even non-member of the SQA team is clearly defined. The reason for this is very simple: when everyone knows their role and boundaries, there is no overlapping of responsibilities and everyone could concentrate on their roles.

But SQA Planning is not only a document that tells who gets to do the specific task. The stages in are also detailed. The whole SQA team will be very busy once the actual testing starts but with SQA, everyone's work is clearly laid out. Through planning, the actual state of the application testing is known. Again in smaller businesses, the planning maybe limited to the phase of the application testing but when outlined for corporations, the scenario changes and only through planning that everyone will know where they are and where they are going in terms of **SQA.**

SQA Planning is not just a simple document where objectives are written and stages are clearly stated. Because of the need to standardize software development ensuring the limitation of error, a scientific approach is recommended in developing an SQA plan. Certain standards such as IEEE Std 730 or 983.

2.5.1 SQA Plan Content

An SQA Plan is detailed description of the project and its approach for testing. Going with the standards, an SQA Plan is divided into four sections:

- Software Quality Assurance Plan for Software Requirements;
- Software Quality Assurance Plan for Architectural Design;
- Software Quality Assurance Plan for Detailed Design and Production and;
- Software Quality Assurance Plan for Transfer

In the first phase, the SQA team should write in detail the activities related for software requirements. In this stage, the team will be creating steps and stages on how they will analyze the software requirements. They could refer to additional documents to ensure the plan works out.

The second stage of SQA Plan or the SQAP for AD (Architectural Design) the team should analyze in detail the preparation of the development team for detailed build-up. This stage is a rough

representation of the program but it still has to go through rigorous scrutiny before it reaches the next stage.

The third phase which tackles the quality assurance plan for detailed design and actual product is probably the longest among phases. The SQA team should write in detail the tools and approach they will be using to ensure that the produced application is written according to plan. The team should also start planning on the transfer phase as well.

The last stage is the QA plan for transfer of technology to the operations. The SQA team should write their plan on how they will monitor the transfer of technology such as training and support.

2.6 SQA LIFECYCLE STANDARDS

Software Quality Assurance procedures have finally been standardized and have been virtually perfected after years of planning on how to perfect the application standardization. Through experience, the company was able to place in writing how to develop a plan for software development. Because it has been standardized, the application that was developed using SQA could be recognized worldwide because it has been made according to the standards. Along with the standards, the metrics are also standardized. More than anything else written in the report, the clients who are looking for an efficient application looks for the numbers more than anything else. Metrics will tell whether the application has been developed according to plan and could perform when released or sold to their intended users.

SQA Standards

IEEE (Institute of Electric and Electronic Engineers) – This standard was established by the organization with the same name. This organization was established in 1963 and the IEEE standards for software development starting in 1986. There are two types of IEEE standards for Software Quality Application: the Standard 730-1989 which was developed in 1989 and the ANSI/IEEE Standard 983 – 1986 which was the original version developed in 1986. IEEE is very popular especially for SQA Planning and development.

ISO (International Standards Organization) – One of the oldest standardization organizations in the world, ISO were established in 1947 and have established itself to be the standardized company not only in software development but also in business plans. Because it was internationally recognized it has become a powerful standard for different business uses.

DOD (US Department of Defense) – The government has also developed their own standardization scheme especially for developing technology. They have evolved from ISO 9000 and developed a

specialized standard on their own. There are currently two DOD standards: the MIL-STD-961E which refers to the program specific standards and the MIL-STD-962D which stands for general standardization. Both of these formats were used applied by the DOD starting August 1, 2003.

ANSI (American National Standards Institute) – Working with US-based companies, ANSI has become the bridge of small US based companies to international standards so that they could achieve international recognition. ANSI covers almost anything in the country, products technology and application. ANSI ensures that the products developed in the US could be used in other countries as well.

IEC (International Electro technical Commission) – Stated June of 1906, the commission has dedicated itself to the standardization of electrical materials and its development. Today it is usually associated with ISO since it has become a part of technical industries. IEC is known for standardizing electronics through the International Electro technical Vocabulary which is used by all electronic industries until today.

EIA (Electronic Industries Alliance) - EIA is a coming together of different electronic manufacturers in US. This organization set the standards for electronic products for the country and has been accepted by thousands of companies worldwide.

SQA Management Standards

Aside from internationally recognized SQA standards, there are specific standards that were developed to cater specifically for the management of software development:

ISO 9001 and ISO 9000-3 – These certifying organizations was established specifically for software development. This standard encourages leadership and could be integrated continuously even when the product has been developed and released to its users. Good supplier relations are also emphasized as technology is not only developed in-house.

SW-CMM (Software Capability Maturity Model) – Developed in 1980, SW-CMM has become the standards for large scale software development companies. It has drawn support because this development model was established by the developers for the developers. It believes in quantitative methods to develop and maintain productivity. SW-CMM has a five-level model to gauge the applications maturity and establish a detailed plan to further enhance them. The best draw so far of SW-CMM is that it does not care about SDLC model, tool and documentation standard, promoting creativity for software development.

ISO/IEC 15504 Process Assessment Model and SPICE (Simulation Program with Integrated Circuit Emphasis) – Aiming for international acceptance, this type of SQA supports a specific type of testing standard

for a better application. Called SPICE, this application could test each part of the application. SPICE is also used to asses the performance of circuits in electronic products.

2.4 - 2.5 Check Your Progress

I. Fill in the Blanks

1. _____ team should analyze in detail the preparation of the development team for detailed build-up.

2. There are two types of IEEE standards for Software Quality Application: the _____which was developed in 1989 and the _____which was developed in 1986.

2.7 SUMMARY

SQA has benefits that will launch the company towards the global standard it needs.SQA should also be utilized at the same way to its full extent.SQA is needed to trace the errors, find the cost affiant, to give flexible solutions and better customer service. SQA team has to follow certain principles like Feedback, Focus on Critical Factor, Multiple Objectives, Evolution, and Quality Control. Motivation, Process Improvement, Persistence, Different Effects of SQA and Result focused.SQA is composed of a variety of tasks linked with 2 dissimilar constituencies: SQA group that has responsibility for quality assurance planning, record keeping, oversight, analysis and reporting and the other are software engineers, who do technical work. The SQA team should ensure that the expected metrics will be posted. At the same time, the SQA team should also select the right tool to gauge the application. Notable SQA tools are WinRunner, LoadRunner, QuickTest Professional, Mercury TestDirector, Silktest and Bugzilla, Application Center Test, OpenSTA, QARun. Selecting the Right Quality Model and Right SQA tool is responsibility of SQA team. SQA management plans involve SQAP, SPMP, SCMP, SVMP. SQA standards are IEEE, ISO, DOD,ANSI, ICE, EIA. SQA Management Standards are ISO, SW-CMM, ISO/IEC.

2.8 CHECK YOUR PROGRESS- ANSWERS

2.2 & 2.3

1. False
2. True

3. False
4. False
5. True

2.4 & 2.5

1. SQAP for AD (Architectural Design)
2. Standard 730-1989 and ANSI/IEEE Standard 983 – 1986

2.9 QUESTIONS FOR SELF - STUDY

1. Explain the principles of Software Quality Assurance.
2. Write a note of software's and tools used for software quality assurance.
3. Explain the four plans involved in Software Management.
4. Explain the organisations involved in SQA standards.
5. Explain the organisations involved in SQA Management standards.

❁❁❁

NOTES

NOTES

Chapter 3

Software Reliability

3.0 OBJECTIVES

Dear friends, after studying this chapter you will able to –

- Discuss the Reliability Measures
- Discuss the Reliability Models

3.1 INTRODUCTION

According to ANSI, Software Reliability is defined as: the probability of failure-free software operation for a specified period of time in a specified environment. Although Software Reliability is defined as a probabilistic function, and comes with the notion of time, we must note that, different from traditional Hardware Reliability, Software Reliability is not a direct function of time. Electronic and mechanical parts may become "old" and wear out with time and usage, but software will not rust or wear-out during its life cycle. Software will not change over time unless intentionally changed or upgraded.

3.2 RELIABILITY MEASURES

Software Reliability is an important to attribute of software quality, together with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve, because the complexity of software tends to be high. While any system with a high degree of complexity, including software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the rapid growth of system size and ease of doing so by upgrading the software. For example, large next-generation aircraft will have over one million source lines of software on-board; next-generation air traffic control

systems will contain between one and two million lines; the upcoming international Space Station will have over two million lines on-board and over ten million lines of ground support software; several major life-critical defence systems will have over five million source lines of software. While the complexity of software is inversely related to software reliability, it is directly related to other important factors in software quality, especially functionality, capability, etc. Emphasizing these features will tend to add more complexity to software.

Software failure mechanisms

Software failures may be due to errors, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. While it is tempting to draw an analogy between Software Reliability and Hardware Reliability, software and hardware have basic differences that make them different in failure mechanisms. Hardware faults are mostly *physical faults*, while software faults are *design faults*, which are harder to visualize, classify, detect, and correct. Design faults are closely related to fuzzy human factors and the design process, which we don't have a solid understanding. In hardware, design faults may also exist, but physical faults usually dominate. In software, we can hardly find a strict corresponding counterpart for "manufacturing" as hardware manufacturing process, if the simple action of uploading software modules into place does not count. Therefore, the quality of software will not change once it is uploaded into the storage and start running. Trying to achieve higher reliability by simply duplicating the same software modules will not work, because design faults cannot be masked off by voting.

A partial list of the distinct characteristics of software compared to hardware is listed below:

- Failure cause: Software defects are mainly design defects.
- Wear-out: Software does not have energy related wear-out phase. Errors can occur without warning.
- Repairable system concept: Periodic restarts can help fix software problems.
- Time dependency and life cycle: Software reliability is not a function of operational time.
- Environmental factors: Do not affect Software reliability, except it might affect program inputs.
- Reliability prediction: Software reliability cannot be predicted from any physical basis, since it depends completely on human factors in design.

- Redundancy: Cannot improve Software reliability if identical software components are used.
- Interfaces: Software interfaces are purely conceptual other than visual.
- Failure rate motivators: Usually not predictable from analyses of separate statements.
- Built with standard components: Well-understood and extensively-tested standard parts will help improve maintainability and reliability. But in software industry, we have not observed this trend. Code reuse has been around for some time, but to a very limited extent. Strictly speaking there are no standard parts for software, except some standardized logic structures.

The bathtub curve for Software Reliability
Over time, hardware exhibits the failure characteristics shown in Figure 1, known as the bathtub curve. Period A, B and C stands for burn-in phase, useful life phase and end-of-life phase. A detailed discussion about the curve can be found in the topic Traditional Reliability.

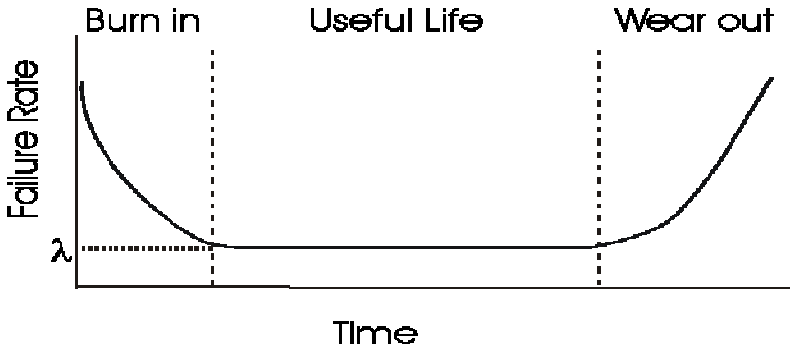


Figure 1. Bath tub curve for hardware reliability

Software reliability, however, does not show the same characteristics similar as hardware. A possible curve is shown in Figure 2 if we projected software reliability on the same axes. There are two major differences between hardware and software curves. One difference is that in the last phase, software does not have an increasing failure rate as hardware does. In this phase, software is approaching obsolescence; there is no motivation for any upgrades or changes to the software. Therefore, the failure rate will not change. The second difference is that in the useful-life phase, software will experience a drastic increase in failure rate each time an upgrade is made. The failure rate levels off gradually, partly because of the defects found and fixed after the upgrades.

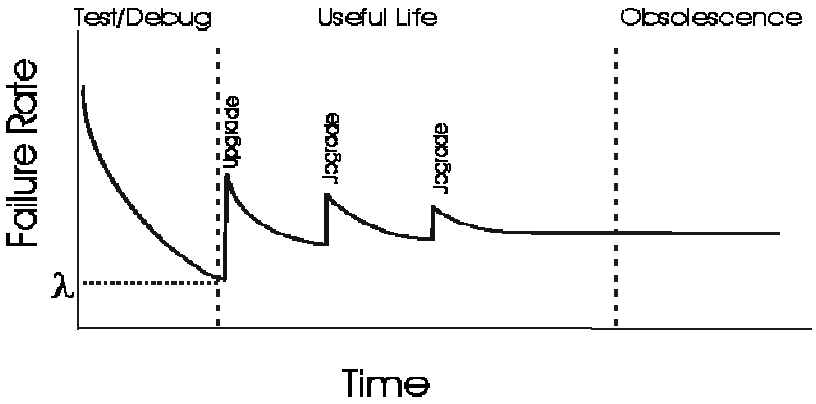


Figure 2. Revised bathtub curve for software reliability

The upgrades in Figure 2 imply feature upgrades, not upgrades for reliability. For feature upgrades, the complexity of software is likely to be increased, since the functionality of software is enhanced. Even bug fixes may be a reason for more software failures, if the bug fix induces other defects into software. For reliability upgrades, it is possible to incur a drop in software failure rate, if the goal of the upgrade is enhancing software reliability, such as a redesign or reimplementation of some modules using better engineering approaches, such as clean-room method.

A proof can be found in the result from Ballista project, robustness testing of off-the-shelf software Components. Figure 3 shows the testing results of fifteen POSIX compliant operating systems. From the graph we see that for QNX and HP-UX, robustness failure rate increases after the upgrade. But for SunOS, IRIX and Digital UNIX, robustness failure rate drops when the version numbers go up. Since software robustness is one aspect of software reliability, this result indicates that the upgrade of those systems shown in Figure 3 should have incorporated reliability upgrades.

Available tools, techniques, and metrics

Since Software Reliability is one of the most important aspects of software quality, Reliability Engineering approaches are practiced in software field as well. *Software Reliability Engineering* (SRE) is the quantitative study of the operational behavior of software-based systems with respect to user requirements concerning reliability.

3.3 SOFTWARE RELIABILITY MODELS

A proliferation of software reliability models have emerged as people try to understand the characteristics of how and why software fails, and try to quantify software reliability. Over 200 models have been

developed since the early 1970s, but how to quantify software reliability still remains largely unsolved. Interested readers may refer to as many models as there are and many more emerging, none of the models can capture a satisfying amount of the complexity of software; constraints and assumptions have to be made for the quantifying process. Therefore, there is no single model that can be used in all situations. No model is complete or even representative. One model may work well for a set of certain software, but may be completely off track for other kinds of problems.

Most software models contain the following parts: assumptions, factors, and a mathematical function that relates the reliability with the factors. The mathematical function is usually higher order exponential or logarithmic.

Software modelling techniques can be divided into two subcategories: prediction modelling and estimation modelling. Both kinds of modelling techniques are based on observing and accumulating failure data and analyzing with statistical inference. The major difference of the two models is shown in Table 1.

ISSUES	PREDICTION MODELS	ESTIMATION MODELS
DATA REFERENCE	Uses historical data	Uses data from the current software development effort
WHEN USED IN DEVELOPMENT CYCLE	Usually made prior to development or test phases; can be used as early as concept phase	Usually made later in life cycle(after some data have been collected); not typically used in concept or development phases
TIME FRAME	Predict reliability at some future time	Estimate reliability at either present or some future time

Table 1. Difference between software reliability prediction models and software reliability estimation models

3.3.1 Software Reliability Metrics

Measurement is commonplace in other engineering field, but not in software engineering. Though frustrating, the quest of quantifying software reliability has never ceased. Until now, we still have no good way of measuring software reliability.

Measuring software reliability remains a difficult problem because we don't have a good understanding of the nature of software. There is no clear definition to what aspects are related to software reliability. We cannot find a suitable way to measure software reliability, and most of the aspects related to software reliability. Even the most obvious product metrics such as software size have not uniform definition.

It is tempting to measure something related to reliability to reflect the characteristics, if we cannot measure reliability directly. The current practices of software reliability measurement can be divided into four categories:

- **Product metrics**

Software size is thought to be reflective of complexity, development effort and reliability. Lines Of Code (LOC), or LOC in thousands (KLOC), is an intuitive initial approach to measuring software size. But there is not a standard way of counting. Typically, source code is used(SLOC, KSLOC) and comments and other non-executable statements are not counted. This method cannot faithfully compare software not written in the same language. The advent of new technologies of code reuses and code generation technique also cast doubt on this simple method.

Function point metric is a method of measuring the functionality of a proposed software development based upon a count of inputs, outputs, master files, inquires, and interfaces. The method can be used to estimate the size of a software system as soon as these functions can be identified. It is a measure of the functional complexity of the program. It measures the functionality delivered to the user and is independent of the programming language. It is used primarily for business systems; it is not proven in scientific or real-time applications.

Complexity is directly related to software reliability, so representing complexity is important. Complexity-oriented metrics is a method of determining the complexity of a program’s control structure, by simplifies the code into a graphical representation. Representative metric is McCabe's Complexity Metric.

Test coverage metrics are a way of estimating fault and reliability by performing tests on software products, based on the assumption that software reliability is a function of the portion of software that has been successfully verified or tested. Detailed discussion about

various software testing methods can be found in topic Software Testing.

- **Project management metrics**

  Researchers have realized that good management can result in better products. Research has demonstrated that a relationship exists between the development process and the ability to complete projects on time and within the desired quality objectives. Costs increase when developers use inadequate processes. Higher reliability can be achieved by using better development process, risk management process, configuration management process, etc.

- **Process metrics**

  Based on the assumption that the quality of the product is a direct function of the process, process metrics can be used to estimate, monitor and improve the reliability and quality of software. ISO-9000 certification, or "quality management standards", is the generic reference for a family of standards developed by the International Standards Organization (ISO).

- **Fault and failure metrics**

  The goal of collecting fault and failure metrics is to be able to determine when the software is approaching failure-free execution. Minimally, both the number of faults found during testing (i.e., before delivery) and the failures (or other problems) reported by users after delivery are collected, summarized and analyzed to achieve this goal. Test strategy is highly relative to the effectiveness of fault metrics, because if the testing scenario does not cover the full functionality of the software, the software may pass all tests and yet be prone to failure once delivered. Usually, failure metrics are based upon customer information regarding failures found after release of the software. The failure data collected is therefore used to calculate failure density, Mean Time Between Failures (MTBF) or other parameters to measure or predict software reliability.

Software Reliability Improvement Techniques

Good engineering methods can largely improve software reliability. Before the deployment of software products, testing, verification and validation are necessary steps. Software testing is heavily used to trigger, locate and remove software defects. Software testing is still in its infant stage; testing is crafted to suit specific needs in various software development projects in an ad-hoc manner. Various analysis tools such as trend analysis, fault-tree analysis, Orthogonal Defect classification and formal methods, etc, can also be used to minimize the possibility of defect occurrence after release and therefore improve software reliability.

After deployment of the software product, field data can be gathered and analyzed to study the behavior of software defects. Fault tolerance or fault/failure forecasting techniques will be helpful techniques and guide rules to minimize fault occurrence or impact of the fault on the system.

Measure	Metrics
1. Customer satisfaction index	Number of system enhancement requests per year Number of maintenance fix requests per year User friendliness: call volume to customer service hotline User friendliness: training time per new user Number of product recalls or fix releases (software vendors) Number of production re-runs (in-house information systems groups)
2. Delivered defect quantities	Normalized per function point (or per LOC) At product delivery (first 3 months or first year of operation) Ongoing (per year of operation) By level of severity By category or cause, e.g.: requirements defect, design defect, code defect, documentation/on-line help defect, defect introduced by fixes, etc.
3. Responsiveness (turnaround time) to users	Turnaround time for defect fixes, by level of severity Time for minor vs. major enhancements; actual vs. planned elapsed time (by customers) in the first year after product delivery
7. Complexity of delivered product	McCabe's cyclomatic complexity counts across the system Halstead's measure Card's design complexity measures Predicted defects and maintenance costs, based on complexity measures
8. Test coverage	Breadth of functional coverage Percentage of paths, branches or conditions that were actually tested Percentage by criticality level: perceived level of risk of paths The ratio of the number of detected faults to the number of predicted faults.
9. Cost of defects	Business losses per defect that occurs during operation Business interruption costs; costs of work-around Lost sales and lost goodwill Litigation costs resulting from defects Annual maintenance cost (per function point) Annual operating cost (per function point) Measurable damage to your boss's career
10. Costs of quality activities	Costs of reviews, inspections and preventive measures Costs of test planning and preparation Costs of test execution, defect tracking, version and change control Costs of diagnostics, debugging and fixing Costs of tools and tool support Costs of tools and tool support Costs of test case library maintenance Costs of testing & QA education

	associated with the product Costs of monitoring and oversight by the QA organization (if separate from the development and test organizations)
11. Re-work	Re-work effort (hours, as a percentage of the original coding hours) Re-worked LOC (source lines of code, as a percentage of the total delivered LOC) Re-worked software components (as a percentage of the total delivered components)
12. Reliability	Availability (percentage of time a system is available, versus the time the system is needed to be available) Mean time between failure (MTBF) Mean time to repair (MTTR) Reliability ratio (MTBF / MTTR) Number of product recalls or fix releases Number of production re-runs as a ratio of production runs

3.2 - 3.3 Check Your Progress

I. Fill in the Blanks

1. Lines of Code _____ or _____ is an intuitive initial approach to measuring software size.
2. Software modelling techniques can be divided into two subcategories: _____ and _____.
3. _____ is the quantitative study of the operational behaviour of software-based systems with respect to user requirements concerning reliability.
4. Hardware faults are mostly *physical faults*, while software faults are _____.
5. Before the deployment of software products, _____, _____ and _____are necessary steps.

3.4 SUMMARY

Software Reliability is defined as a probabilistic function, and comes with the notion of time, we must note that, different from traditional Hardware Reliability, Software Reliability is not a direct function of time. Software Reliability is an important to attribute of software quality, together with functionality, usability, performance, service ability, capability, installability, maintainability, and documentation. While the complexity of software is inversely related to software reliability, it is

directly related to other important factors in software quality, especially functionality, capability, etc. Emphasizing these features will tend to add more complexity to software. Software failures may be due to errors, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems. Hardware faults are mostly physical faults, while software faults are design faults. Bathtub curve for hardware reliability and Bathtub curve for software reliability can be used as SRE. Most software models contain the following parts: assumptions, factors, and a mathematical function that relates the reliability with the factors. Software modelling techniques can be divided into two subcategories: prediction modelling and estimation modelling. The current practices of software reliability measurement can be divided into four categories : Product metrics, Project management metrics, Process metrics, Fault and failure metrics.

3.5 CHECK YOUR PROGRESS- ANSWERS

3.2 - 3.3

1. LOC and KLOC
2. Prediction modelling and Estimation modelling.
3. Software Reliability Engineering (SRE)
4. design faults,
5. testing, verification and validation

3.6 QUESTIONS FOR SELF- STUDY

1. Explain bathtub curve for software reliability and Hardware reliability.
2. Write categories for software reliability matrix.
3. Write the difference between software reliability prediction models and software reliability estimation models.

NOTES

NOTES

CHAPTER 4

Verification & Validation

4.0 OBJECTIVES

Dear friends,

After studying this chapter you will be able to -

- Discuss Verification & Validation Planning
- Explain Software inspections
- Discuss Automated static Analysis
- Discuss Clean room Software Development

4.1 INTRODUCTION

A perfect software product is built when every step is taken with full consideration that 'A right product is developed in a right manner'. 'Software Verification & Validation' is one such model, which helps the system designers and test engineers to confirm that a right product is build right way throughout the development process and improve the quality of the software product.

'Verification & Validation Model' makes it sure that, certain rules are followed at the time of development of a software product and also makes it sure that the product that is developed fulfills the required specifications. This reduces the risk associated with any software project up to certain level by helping in detection and correction of errors and mistakes, which are unknowingly done during the development process.

4.2 VERIFICATION AND VALIDATION PLANNING

What is Verification?

The standard definition of Verification goes like this: "Are we building the product RIGHT?" i.e. Verification is a process that makes it sure that the software product is developed the right way. The software should confirm to its predefined specifications, as the product development goes through different stages, an analysis is done to ensure that all required specifications are met.

Methods and techniques used in the Verification and Validation shall be designed carefully, the planning of which starts right from the beginning of the development process. The Verification part of 'Verification and Validation Model' comes before Validation, which incorporates Software inspections, reviews, audits, walkthroughs, buddy checks etc. in each phase of verification (every phase of Verification is a phase of the Testing Life Cycle).

During the Verification, the work product (the ready part of the Software being developed and various documentations) is reviewed/examined personally by one ore more persons in order to find and point out the defects in it. This process helps in prevention of potential bugs, which may cause in failure of the project.

Few terms involved in Verification:

Inspection:

Inspection involves a team of about 3-6 people, led by a leader, which formally reviews the documents and work product during various phases of the product development life cycle. The work product and related documents are presented in front of the inspection team, the member of which carry different interpretations of the presentation. The bugs that are detected during the inspection are communicated to the next level in order to take care of them.

Walkthroughs:

Walkthrough can be considered same as inspection without formal preparation (of any presentation or documentations). During the walkthrough meeting, the presenter/author introduces the material to all the participants in order to make them familiar with it. Even when the walkthroughs can help in finding potential bugs, they are used for knowledge sharing or communication purpose.

Buddy Checks:

This is the simplest type of review activity used to find out bugs in a work product during the verification. In buddy check, one person goes through the documents prepared by another person in order to find out if that person has made mistake(s) i.e. to find out bugs which the author couldn't find previously.

The activities involved in Verification process are: Requirement Specification verification, Functional design verification, internal/system design verification and code verification (these phases can also subdivided further). Each activity makes it sure that the product is developed right way and every requirement; every specification, design code etc. is verified!

What is Validation?

Validation is a process of finding out if the product being built is right? i.e. whatever the software product is being developed, it should do what the user expects it to do. The software product should functionally do what it is supposed to, it should satisfy all the functional requirements set by the user. Validation is done during or at the end of the development process in order to determine whether the product satisfies specified requirements.

Validation and Verification processes go hand in hand, but visibly Validation process starts after Verification process ends (after coding of the product ends). Each Verification activity (such as Requirement Specification Verification, Functional design Verification etc.) has its corresponding Validation activity (such as Functional Validation/Testing, Code Validation/Testing, System/Integration Validation etc.).

All types of testing methods are basically carried out during the Validation process. Test plan, test suits and test cases are developed, which are used during the various phases of Validation process. The phases involved in Validation process are: Code Validation/Testing, Integration Validation/Integration Testing, Functional Validation / Functional Testing, and System/User Acceptance Testing/Validation.

Terms used in Validation process:

Code Validation / Testing:

Developers as well as testers do the code validation. Unit Code Validation or Unit Testing is a type of testing, which the developers conduct in order to find out any bug in the code unit/module developed by them. Code testing other than Unit Testing can be done by testers or developers.

Integration Validation/Testing:

Integration testing is carried out in order to find out if different (two or more) units/modules co-ordinate properly. This test helps in finding out if there is any defect in the interface between different modules.

Functional Validation/Testing:
This type of testing is carried out in order to find if the system meets the functional requirements. In this type of testing, the system is validated for its functional behavior. Functional testing does not deal with internal coding of the project, in stead, it checks if the system behaves as per the expectations.

User Acceptance Testing or System Validation:

In this type of testing, the developed product is handed over to the user/paid testers in order to test it in real time scenario. The product is validated to find out if it works according to the system specifications and satisfies all the user requirements. As the user/paid testers use the software, it may happen that bugs that are yet undiscovered, come up, which are communicated to the developers to be fixed. This helps in improvement of the final product.

4.3 SOFTWARE INSPECTIONS

Inspection in software engineering refers to peer review of any work product by trained individuals who look for defects using a well defined process. An inspection might also be referred to as a Fagan inspection after Michael Fagan, the creator of a very popular software inspection process.

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

Inspection Process

The inspection process was developed by Michael Fagan in the mid-1970s and it has later been extended and modified. The process should have entry criteria that determine if the inspection process is ready to begin. This prevents unfinished work products from entering the inspection process. The entry criteria might be a checklist including items such as "The document has been spell-checked".

The stages in the inspections process are: Planning, Overview meeting, Preparation, Inspection meeting, Rework and Follow-up. The Preparation, Inspection meeting and Rework stages might be iterated.

- **Planning :** The inspection is planned by the moderator.

- Overview meeting: The author describes the background of the work product.
- **Preparation** identify possible defects**:** Each inspector examines the work product to.
- Inspection meeting: During this meeting the reader reads through the work product, part by part and the inspectors point out the defects for every part.
- **Rework :** The author makes changes to the work product according to the action plans from the inspection meeting.

- **Follow-up :** The changes by the author are checked to make sure everything is correct.

The process is ended by the moderator when it satisfies some predefined exit criteria.

Inspection Roles

During an inspection the following roles are used.

- Author: The person who created the work product being inspected.
- Moderator: This is the leader of the inspection. The moderator plans the inspection and coordinates it.
- Reader: The person reading through the documents, one item at a time. The other inspectors then point out defects.
- Recorder/Scribe: The person that documents the defects that are found during the inspection.
- Inspector: The person that examines the work product to identify possible defects.

Related Inspection types

Code review

A code review can be done as a special kind of inspection in which the team examines a sample of code and fixes any defects in it. In a code review, a defect is a block of code which does not properly implement its requirements, which does not function as the programmer intended, or which is not incorrect but could be improved (for example, it could be made more readable or its performance could be improved). In addition to helping teams find and fix bugs, code reviews are useful for both cross-training programmers on the code being reviewed and for helping junior developers learn new programming techniques.

Peer Reviews

Peer Reviews are considered an industry best-practice for detecting software defects early and learning about software artifacts. Peer Reviews are composed of software walkthroughs and software inspections and are integral to software product engineering activities. A

collection of coordinated knowledge, skills, and behaviors facilitates the best possible practice of Peer Reviews. The elements of Peer Reviews include the structured review process, standard of excellence product checklists, defined roles of participants, and the forms and reports.

Software inspections are the most rigorous form of Peer Reviews and fully utilize these elements in detecting defects. Software walkthroughs draw selectively upon the elements in assisting the producer to obtain the deepest understanding of an artifact and reaching a consensus among participants. Measured results reveal that Peer Reviews produce an attractive return on investment obtained through accelerated learning and early defect detection. For best results, Peer Reviews are rolled out within an organization through a defined program of preparing a policy and procedure, training practitioners and managers, defining measurements and populating a database structure, and sustaining the roll out infrastructure.

Check Your Progress 4.2 & 4.3

Fill in the Blanks

1. _____in software engineering refers to peer review of any work product by trained individuals.
2. _____ is a process that makes it sure that the software product is developed the right way.

4.4 AUTOMATED STATIC ANALYSIS

Software vulnerabilities are added into programs during its development. Architectural flaws are introduced during planning and design, while implementation faults are created during coding. Penetration testing is often used to detect these vulnerabilities. This approach is expensive because it is performed late in development and any correction would increase lead-time. An alternative would be to detect and correct vulnerabilities in the phase of development where they are the least expensive to correct and detect. Source code audits have often been suggested and used to detect implementations vulnerabilities. However, manual audits are time consuming and require extended expertise to be efficient.

A static code analysis tool could achieve the same results as a manual audit but at fraction of the time. Through a set of cases studies and experiments at Ericsson AB, this thesis investigates the technical capabilities and limitations of using a static analysis tool as an early vulnerability detector. The investigation is extended to studying the human factor by examining how the developers interact and use the

static analysis tool. The contributions of this thesis include the identification of the tools capabilities so that further security improvements can focus on other types of vulnerabilities. By using static analysis early in development possible cost saving measures are identified. Additionally, the thesis presents the limitations of static code analysis. The most important limitation being the incorrect warnings that are reported by static analysis tools. In addition, a development process overhead was deemed necessary to successfully use static analysis in an industry setting.

We categorize the output or lack of it, from static analysis tools in the following four groups:

- ***False Positive:*** Warnings that do not cause a fault in the software or state and untrue fact. These are often caused be either weak verification or incomplete/weak checkers.
- ***True Positive:*** Correct reports of faults within the software. However, the fault does not allow a software user to create a failure that would result into any of the four security consequences stated in section 1.1.2.
- ***Security Positive:*** Warnings that are correct and can by exploited into any of the four effects in section 1.1.2.
- ***False Negative:*** Known vulnerabilities that the static analysis tool did not report. Either because the analysis lacked the precision required to detect them or because there are no rules or checks that look for the particle vulnerability.

4.5 CLEAN ROOM SOFTWARE DEVELOPMENT

The Cleanroom Software Engineering process is a software development process intended to produce software with a certifiable level of reliability. The Cleanroom process was originally developed by Harlan Mills and several of his colleagues including Alan Hevner at IBM. The focus of the Cleanroom process is on defect prevention, rather than defect removal. The name *Cleanroom* was chosen to evoke the cleanrooms used in the electronics industry to prevent the introduction of defects during the fabrication of semiconductors. The Cleanroom process first saw use in the mid to late 80s. Demonstration projects within the military began in the early 1990s. Recent work on the Cleanroom process has examined fusing Cleanroom with the automated *verification capabilities provided by specifications expressed in CSP.*

Central principles

The basic principles of the Cleanroom process are Software development based on formal methods.

Cleanroom development makes use of the Box Structure Method to specify and design a software product. Verification that the design correctly implements the specification is performed through team review.

Incremental implementation under statistical quality control

Cleanroom development uses an iterative approach, in which the product is developed in increments that gradually increase the implemented functionality. The quality of each increment is measured against pre-established standards to verify that the development process is proceeding acceptably. A failure to meet quality standards results in the cessation of testing for the current increment, and a return to the design phase.

Statistically sound testing

Software testing in the Cleanroom process is carried out as a statistical experiment. Based on the formal specification, a representative subset of software input/output trajectories is selected and tested. This sample is then statistically analyzed to produce an estimate of the reliability of the software, and a level of confidence in that estimate.

4.4 & 4.5 Check Your Progress

Fill in the Blanks

1. Automated static analyzers are used to remove software _____ .

2. The _____ process is a software development process intended to produce software with a certifiable level of reliability.

4.6 SUMMARY

Verification is a process that makes it sure that the software product is developed the right way. The software should confirm to its predefined specifications, as the product development goes through different stages, an analysis is done to ensure that all required specifications are met. The Verification part of 'Verification and Validation Model' comes before Validation, which incorporates Software inspections, reviews, audits, walkthroughs, buddy checks etc. in each phase of verification (every phase of Verification is a phase of the Testing Life Cycle). Few terms involved in verification Inspection, Walkthrough and Buddy checks. The software product should functionally do what it is supposed to, it should satisfy all the functional requirements set by the user. Validation is done during or at the end of the development process

in order to determine whether the product satisfies specified requirements. Validation and Verification processes go hand in hand, but visibly Validation process starts after Verification process ends. The goal of the Software inspection is to identify defects. In an inspection, a defect is any part of the work product that will keep an inspector from approving it. The stages in the inspections process are: Planning, Overview meeting, Preparation, Inspection meeting, Rework and Follow-up. The Preparation, Inspection meeting and Rework stages might be iterated. Software vulnerabilities are added into programs during its development. Architectural flaws are introduced during planning and design, while implementation faults are created during coding. Penetration testing is often used to detect these vulnerabilities. This approach is expensive because it is performed late in development and any correction would increase lead-time.

4.7 CHECK YOUR PROGRESS- ANSWERS

4.2 & 4.3

1. Inspection
2. Verification

4.4 & 4.5

1. vulnerabilities
2. Cleanroom Software Engineering

4.8 QUESTIONS FOR SELF – STUDY

1. Explain in detail Cleanroom Software Engineering.
2. Described the terms used in software validations.
3. What is Verifications? Write the verification terms.
4. Write a note on Software Inspection'.
5. Write the difference between Software verification and Software Validation.

NOTES

CHAPTER 5

Software Testing Fundamentals

5.0 OBJECTIVES

Dear Friends,
After studying this Chapter you will be able to -

- Discuss Testing objectives
- Explain how test information flows
- Discuss Testing lifecycle
- Discuss Test Cases

5.1 INTRODUCTION

Testing has various objectives; it has to find out the bugs as early as possible. In testing department test information flows in various forms. Just like software development life cycle, a testing life cycle is there which decides the test activities. A test case is the basic part of any testing activity; properly designed test cases will discover more and more undiscovered errors.

5.2 TESTING OBJECTIVES

- Testing is a process of executing a program with the intent of finding an error.
- A good test is one that has a high probability of finding an as yet undiscovered error.

- A successful test is one that uncovers an as yet undiscovered error.
- The objective is to design tests that systematically uncover different classes of errors and do so with a minimum amount of time and effort.

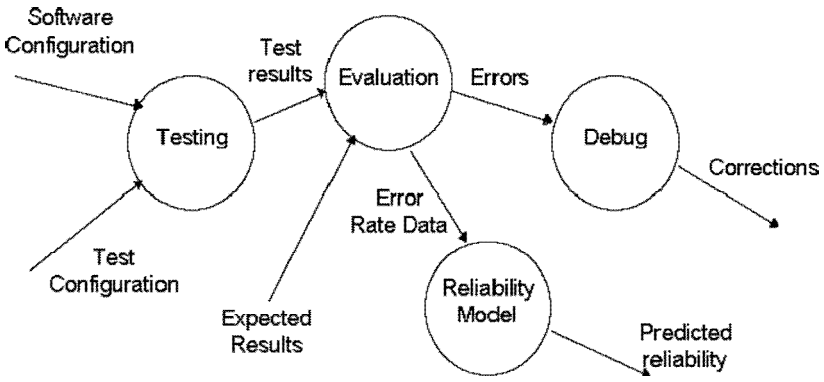
Secondary benefits include:

Demonstrate that software functions appear to be working according to specification. Those performance requirements appear to have been met.

Data collected during testing provides a good indication of software reliability and some indication of software quality.

Testing cannot show the absence of defects, it can only show that software defects are present

5.3 HOW TEST INFORMATION FLOWS



In the above figure:

Software configuration includes a Software Requirements Specification, a Design Specification, and source code.

A test configuration includes a test plan and procedures, test cases, and testing tools. It is difficult to predict the time to debug the code, hence it is difficult to schedule.

5.4 TESTING LIFECYCLE

Software testing *life cycle* identifies what test activities to carry out and when (what is the best time) to accomplish those test activities. Even though testing differs between organizations, there is a testing life cycle.

Software Testing Life Cycle consists of six (generic) phases:

- Test Planning,
- Test Analysis,
- Test Design,
- Construction and verification,
- Testing Cycles,
- Final Testing and Implementation and
- Post Implementation.

Software testing has its own life cycle that intersects with every stage of the SDLC. The basic requirements in software testing life cycle is to control/deal with software testing – Manual, Automated and Performance.

- **Test Planning**

  This is the phase where Project Manager has to decide what things need to be tested, do I have the appropriate budget etc. Naturally proper planning at this stage would greatly reduce the risk of low quality software. This planning will be an ongoing process with no end point.

  Activities at this stage would include preparation of high level test plan-(according to IEEE test plan template The Software Test Plan (STP) is designed to prescribe the scope, approach, resources, and schedule of all testing activities. The plan must identify the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the resources and schedule required to complete testing, and the risks associated with the plan.). Almost all of the activities done during this stage are included in this software test plan and revolve around a test plan.

- **Test Analysis**

  Once test plan is made and decided upon, next step is to delve little more into the project and decide what types of testing should be carried out at different stages of SDLC, do we need or plan to automate, if yes then when the appropriate time to automate is, what type of specific documentation I need for testing.

  Proper and regular meetings should be held between testing teams, project managers, and development teams, Business Analysts to check the progress of things which will give a fair idea of the movement of the project and ensure the completeness of the test plan created in the planning phase, which will further help in enhancing the right testing strategy created earlier. We will start creating test case formats and test cases itself.

  In this stage we need to develop Functional validation matrix based

on Business Requirements to ensure that all system requirements are covered by one or more test cases, identify which test cases to automate, begin review of documentation, i.e. Functional Design, Business Requirements, Product Specifications, Product Externals etc. We also have to define areas for Stress and Performance testing.

- **Test Design**

  Test plans and cases which were developed in the analysis phase are revised. Functional validation matrix is also revised and finalized. In this stage risk assessment criteria is developed. If you have thought of automation then you have to select which test cases to automate and begin writing scripts for them. Test data is prepared. Standards for unit testing and pass / fail criteria are defined here. Schedule for testing is revised (if necessary) & finalized and test environment is prepared.

- **Construction and Verification**

  In this phase we have to complete all the test plans, test cases, complete the scripting of the automated test cases, Stress and Performance testing plans needs to be completed. We have to support the development team in their unit testing phase. And obviously bug reporting would be done as when the bugs are found. Integration tests are performed and errors (if any) are reported.

- **Testing Cycles**

  In this phase we have to complete testing cycles until test cases are executed without errors or a predefined condition is reached. Run test cases –> Report Bugs –> revise test cases (if needed) –> add new test cases (if needed) –> bug fixing –> retesting (test cycle 2, test cycle 3….).

- **Final Testing and Implementation**

  In this we have to execute remaining stress and performance test cases, documentation for testing is completed / updated, provide and complete different matrices for testing. Acceptance, load and recovery testing will also be conducted and the application needs to be verified under production conditions.

- **Post Implementation**

  In this phase, the testing process is evaluated and lessons learnt from that testing process are documented. Line of attack to prevent similar problems in future projects is identified. Create plans to improve the processes. The recording of new errors and enhancements is an ongoing process. Cleaning up of test environment is done and test machines are restored to base lines in this stage.

Software Testing Life cycle		
Phase	Activities	Outcome
Planning	Create high level test plan	Test plan, Refined Specification
Analysis	Create detailed test plan, Functional Validation Matrix, test cases	Revised Test Plan, Functional validation matrix, test cases
Design	test cases are revised; select which test cases to automate	revised test cases, test data sets, sets, risk assessment sheet
Construction	scripting of test cases to automate,	test procedures/Scripts, Drivers, test results, Bug reports.
Testing cycles	complete testing cycles	Test results, Bug Reports
Final testing	execute remaining stress and performance tests, complete documentation	Test results and different metrics on test efforts
Post implementation	Evaluate testing processes	Plan for improvement of testing process

5.2, 5.4 Check your progress

Fill in the blanks

1. A successful test is one that uncovers an as yet _____error.
2. In _____ phase where Project Manager has to decide what things need to be tested, do he has the appropriate budget etc
3. _____ phase Evaluate testing processes.

5.5 Test Cases – What it is?, Test Case designing

Test cases are the specific inputs that you'll try and the procedures that you'll follow when you test the software.

Addition Test cases for windows calculator

0+0	should	equal to	0
0+1	should	equal to	1
54+1	should	equal to	55
254+1	should	equal to	255
1022+1	should	equal to	1023

.....
.....

Selecting test cases is the single most important task that software testers do.

Improper selection can result in testing too much , testing too little, or testing the wrong things.

Intelligently weighing the risks and reducing the infinite possibilities to a manageable effective set is where the magic is.

Importance of Test design

- Test cases form the foundation on which to design and develop test scripts
- The depth of testing is proportional to number of test cases
- A principle measure of completeness of test is requirements based coverage based on the number of test cases identified, implemented and /or executed
- The scale of test effort is proportional to number of test cases
- The kinds of test design and development and the resources needed are largely governed by the test cases.

Test Design Essentials

- Test case should cover all features
- There should be balance between all types of test cases
- The test cases should be documented properly

5.6 SUMMARY

The main objective of testing is to find the bugs as early as possible. A successful test finds yet an undiscovered error. Test information flows from various phases in variety of forms. Just like SDLC Software test life cycle identifies what test activities to carry out and when to accomplish those test activities. STLC has phases named Test Planning, Test Analysis, Test Design, Construction and verification, Testing Cycles,

Final Testing and Implementation and Post Implementation. Test cases are the important part of testing activity that consists of the inputs and procedures used to test the software. Selecting test cases is the single most important task that software testers do. Test design is important as it decides the number and type of test cases which has the high probability of finding an error.

5.7 CHECK YOUR PROGRESS - ANSWERS

5.2 - 5.4

1. Undiscovered
2. Test planning
3. Post implementation

5.8 QUESTIONS FOR SELF - STUDY

1. What is the objective of testing?
2. Explain how test information flows using suitable diagram.
3. Describe software testing life cycle.
4. What do you mean by a test case? Explain the importance of test case design.

❁ ❁ ❁

NOTES

Chapter 6

Levels of Testing

6.0 OBJECTIVES

Dear Friends,
After studying this Chapter you will be able to -

- Discuss Unit Testing
- Describe Integration Testing
- Describe System Testing
- Explain Acceptance Testing
- Discuss Alpha testing & Beta testing
- Distinguish between Static vs. Dynamic testing
- Distinguish between Manual vs. Automatic testing
- Discuss Testers workbench
- Explain 11-steps of testing process

6.1 INTRODUCTION

There are various testing levels one of which is unit testing in which the smallest testable part of an application is testing for correctness. In integration testing we check the system when we linking

the various modules. In system testing we check the system as a whole from customers' viewpoint. Acceptance testing tries to check whether the system is acceptable by most of the users. Alpha testing is carried out at developer's site and beta is at customer's site. A Testers workbench is a virtual environment used to verify the correctness or soundness of a design or model. 11 step testing process is a experience based practical approach for solution to a test assignment.

6.2 UNIT TESTING

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing is often automated but it can also be done manually.

Unit testing involves only those characteristics that are vital to the performance of the unit under test. This encourages developers to modify the source code without immediate concerns about how such changes might affect the functioning of other units or the program as a whole. Once all of the units in a program have been found to be working in the most efficient and error-free manner possible, larger components of the program can be evaluated by means of integration testing.

Unit testing can be time-consuming and tedious. It demands patience and thoroughness on the part of the development team. Rigorous documentation must be maintained. Unit testing must be done with an awareness that it may not be possible to test a unit for every input scenario that will occur when the program is run in a real-world environment.

6.3 INTEGRATION TESTING

Integration testing is a systematic technique for constructing the program structure while conducting tests to uncover errors associated with interfacing. The objective is to take unit tested modules and build a program structure that has been dictated by Design Different Integration strategies

There are two approaches in integration testing they are as follows -

Top-down integration is an incremental approach to construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module.

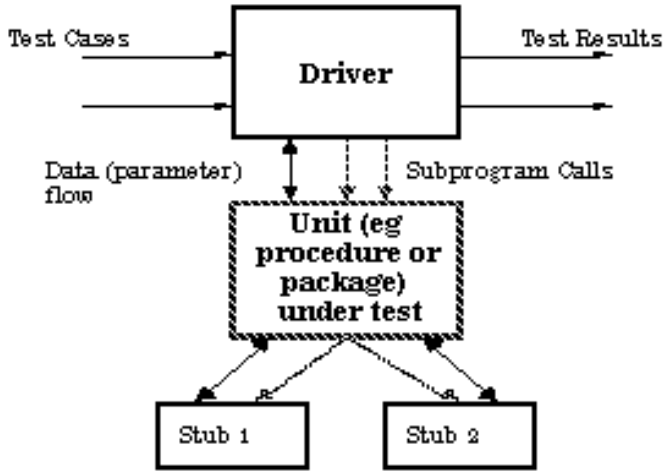
The integration process is performed in a series of five steps:

1. The main control module is used as a test driver, and stubs are substituted for all modules directly subordinate to the main control module.

2. Depending on the integration approach selected (i.e., depth-or breadth first), subordinate stubs are replaced one at a time with actual modules.
3. Tests are conducted as each modules are integrated
4. On completion of each set of tests, another stub is replaced with real module
5. Regression testing may be conducted to ensure that new errors have not been introduced

The process continues from step2 until the entire program structure is built.

Top-down strategy sounds relatively uncomplicated, but in practice, logistical problems arise. The most common of these problems occurs when processing at low levels in the hierarchy is required to adequately test upper levels. Stubs replace low-level modules at the beginning of top-down testing; therefore, no significant data can flow upward in the program structure.



Bottom up Integration

Modules are integrated from the bottom to top, in this approach processing required for modules subordinate to a given level is always available and the needs for subs is eliminated.

A bottom-up integration strategy may be implemented with the following steps:

1. Low-level modules are combined into clusters that perform a specific software sub function.

2. A driver is written to coordinate test case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

As integration moves upward, the need for separate test drivers lessens. In fact, if the top two levels of program structure are integrated top-down, the number of drivers can be reduced substantially and integration of clusters is greatly simplified.

6.4 SYSTEM TESTING

Once the entire system has been built then it has to be tested against the "System Specification" to check if it delivers the features required. It is still developer focussed, although specialist developers known as systems testers are normally employed to do it.

In essence System Testing is not about checking the individual parts of the design, but about checking the system as a whole. In effect it is one giant component.

System testing can involve a number of specialist types of tests to see if all the functional and non- functional requirements have been met. In addition to functional requirements these may include the following types of testing for the non-functional requirements:

Performance -	Are the performance criteria met?
Volume -	Can large volumes of information be handled?
Stress -	Can peak volumes of information be handled?
Documentation -	Is the documentation usable for the system?
Robustness -	Does the system remain stable under adverse circumstances?

There are many others, the needs for which are dictated by how the system is supposed to perform.

6.5 ACCEPTANCE TESTING

Acceptance Testing checks the system against the "Requirements". It is similar to systems testing in that the whole system is checked but the important difference is the change in focus. Systems Testing checks that the system that was specified has been delivered. Acceptance Testing checks that the system delivers what was requested.

The customer and not the developer should always do acceptance testing. The customer knows what is required from the system to achieve value in the business and is the only person qualified to make that judgement.

6.6 ALPHA TESTING & BETA TESTING

If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one. Most software product builders use a process called alpha beta testing to uncover errors that only the end user seems able to find.

The alpha test conducted at the developer's site by a customer software is used in a natural setting with the developer "Looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.

The beta test is conducted at one or more customer sites by the end user(S) of the software. Unlike alpha testing the developer is generally not present; therefore the beta test is "live".

Application of the software in an environment that cannot be controlled by the developer.

The customer records all problems (real/imagined) that are encountered during beta testing and reports these to the developer at regular intervals. Because of problems reported during beta test, the software developer makes modification and then prepares for release of the software product to the entire customer base.

6.2 to 6.6 Check your Progress

Q1. Fill in the blanks

1. _____ testing is a software development process in which the smallest testable parts of an application are individually and independently.
2. _____ type of testing checks if the system delivers the features required.
3. _____ testing checks the system against the Requirements.
4. In Top down integration testing _____ are used.

   A) Stubs b) Drivers
5. In Bottom up integration testing _____ are used.

   A) Stubs b) Drivers

Q2. State whether True or False

1. The alpha test is conducted at the User's site.
2. The beta test is conducted in 'live' environment.
3. Integration testing uncovers errors associated with interfacing.

6.7 STATIC VS. DYNAMIC TESTING

Software can be tested either by running the programs and verifying each step of its execution against expected results or by statically examining the code or the document against its stated requirement or objective. In general, software testing can be divided into two categories, viz. Static and dynamic testing.

Static testing is a non-execution-based testing and carried through by mostly human effort. In static testing, we test, design, code or any document through inspection, walkthroughs and reviews. Many studies show that the single most cost-effective defect reduction process is the classic structural test; the code inspection or walk-through. Code inspection is like proof reading and developers will be benefited in identifying the typographical errors, logic errors and deviations in styles and standards normally followed.

Dynamic testing is an execution based testing technique. Program must be executed to find the possible errors. Here, the program, module or the entire system is executed (run) and the output is verified against the expected result. Dynamic execution of tests is based on specifications of the program, code and methodology.

6.8 MANUAL VS. AUTOMATIC TESTING

Not all tests can be automated and most times is difficult to decide what to automate and what to manually test. Here are some advantages and disadvantages of both methods.

ADVANTAGES	
Automated Testing	Manual Testing
• If you have to run a set of tests repeatedly automation is a huge gain	• If Test Cases have to be run a small number of times it's more likely to perform manual testing
• Helps performing "compatibility testing" - testing the software on different configurations	• It allows the tester to perform more ad-hoc (random testing)
• It gives you the ability to run automation scenarios to perform regressions in a shorter time	• Short term costs are reduced
• It gives you the ability to run regressions on a code that is continuously changing	• The more time tester spends testing a module the grater the odds to find real user bugs

• Can be run simultaneously on different machines thus decreasing testing time	
• Long term costs are reduced	
DISADVANTAGES	
Automated Testing	Manual Testing
• It's more expensive to automate. Initial investments are bigger than manual testing	• Manual tests can be very time consuming
• You cannot automate everything, some tests still have to be done manually	• For every release you must rerun the same set of tests which can be tiresome

6.9 TESTERS WORKBENCH

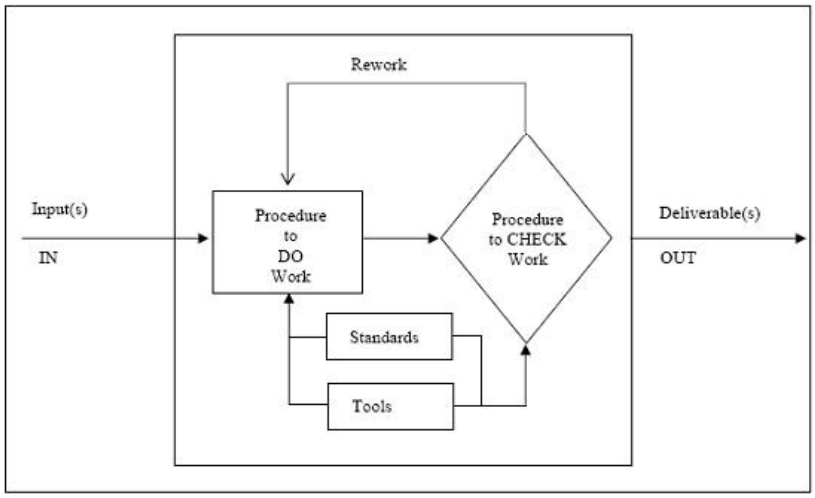
A Testers workbench is a virtual environment used to verify the correctness or soundness of a design or model (e.g., a software product).

The term has its roots in the testing of electronic devices, where an engineer would sit at a lab bench with tools of measurement and manipulation, such as oscilloscopes, multimeters, soldering irons, wire cutters, and so on, and manually verify the correctness of the device under test.

In the context of software or firmware or hardware engineering, a test bench refers to an environment in which the product under development is tested with the aid of a collection of testing tools. Often, though not always, the suite of testing tools is designed specifically for the product under test.

A test bench or testing workbench has four components.

1. INPUT: The entrance criteria or deliverables needed to perform work
2. PROCEDURES TO DO: The tasks or processes that will transform the input into the output
3. PROCEDURES TO CHECK: The processes that determine that the output meets the standards.
4. OUTPUT: The exit criteria or deliverables produced from the workbench



Testers workbench

6.10 11-STEPS OF TESTING PROCESS

We introduce 11 step testing process that will take you through identifying, testing and solving your test assignment. This process is based on experience.

Step 1: Asses Development Plan and Status

This first step is a prerequisite to building the Verification, Validation, and Testing (VV&T)Plan used to evaluate the implemented software solution. During this step, testers challenge the completeness and correctness of the development plan. Based on the extensiveness and completeness of the Project Plan the testers can estimate the amount of resources they will need to test the implemented software solution.

Step 2: Develop the Test Plan

Forming the plan for testing will follow the same pattern as any software planning process. The structure of all plans should be the same, but the content will vary based on the degree of risk the testers perceive as associated with the software being developed.

Step 3: Test Software Requirements

Incomplete, inaccurate, or inconsistent requirements lead to most software failures. The inability to get requirement right during the requirements gathering phase can also increase the cost of implementation significantly. Testers, through verification, must determine that the requirements are accurate, complete, and they do not conflict with another.

Step 4: Test Software Design

This step tests both external and internal design primarily through verification techniques. The testers are concerned that the design will achieve the objectives of the requirements, as well as the design being effective and efficient on the designated hardware.

Step 5: Program (Build) Phase Testing

The method chosen to build the software from the internal design document will determine the type and extensiveness of the testers needed. As the construction becomes more automated, less testing will be required during this phase. However, if software is constructed using the waterfall process, it is subject to error and should be verified. Experience has shown that it is significantly cheaper to identify defects during the construction phase, than through dynamic testing during the test execution step.

Step 6: Execute and Record Result

This involves the testing of code in a dynamic state. The approach, methods, and tools specified in the test plan will be used to validate that the executable code in fact meets the stated software requirements, and the structural specifications of the design.

Step 7: Acceptance Test

Acceptance testing enables users to evaluate the applicability and usability of the software in performing their day-to-day job functions. This tests what the user believes the software should perform, as opposed to what the documented requirements state the software should perform.

Step 8: Report Test Results

Test reporting is a continuous process. It may be both oral and written. It is important that defects and concerns be reported to the appropriate parties as early as possible, so that corrections can be made at the lowest possible cost.

Step 9: The Software Installation

Once the test team has confirmed that the software is ready for production use, the ability to execute that software in a production environment should be tested. This tests the interface to operating software, related software, and operating procedures.

Step 10: Test Software Changes

While this is shown as Step 10, in the context of performing maintenance after the software is implemented, the concept is also applicable to changes throughout the implementation process. Whenever requirements changes, the test plan must change, and the impact of that change on software systems must be tested and evaluate.

Step 11: Evaluate Test Effectiveness

Testing improvement can best be achieved by evaluating the effectiveness of testing at the end of each software test assignment. While this assessment is primarily performed by the testers, it should involve the developers, users of the software, and quality assurance professionals if the function exists in the IT organization.

6.7 - 6.10 Check your Progress

Fill in the blanks

1. If Test Cases have to be run a small number of times it's more likely to perform _____ testing.
2. A _____is a virtual environment used to verify the correctness or soundness of a design or model.
3. _____ testing is an execution based testing technique.

6.11 SUMMARY

Unit testing involves only those characteristics that are vital to the performance of the unit (smallest testable part of an application)under test. Integration testing consists of conducting tests to uncover errors associated with interfacing (interlinking the modules). Stubs are used in top down integration testing to temporarily replace the lower module whereas Driver module is used in bottom up integration testing to temporarily replace the top module of the application under test. System testing is testing the whole application against system specifications while acceptance testing checks the system against requirements. Alpha test is carried out at developer's site and beta testing is at user's site.

Static testing is a non-execution-based testing and carried through by mostly human effort. In dynamic testing Program must be executed to find the possible errors. In manual testing each and every case is run by the tester and in Automated testing specific automated tools are used to test the software. A test bench refers to an environment in which the product under development is tested with the aid of a collection of testing tools

6.12 CHECK YOUR PROGRESS- ANSWERS

6.2 to 6.6

Q1.

1. Unit
2. System
3. Acceptance
4. Stubs
5. Drivers

Q2.

1. False
2. True
3. True

6.7 to 6.10

1. Manual
2. Testers workbench
3. Dynamic

6.13 QUESTIONS FOR SELF - STUDY

1. Write a note on unit testing.
2. What do you mean by integration testing? Explain its different strategies.
3. Explain a) System testing, b) Acceptance testing
4. Differentiate between a) Alpha testing & Beta testing, b) Static & Dynamic testing, c) Manual vs. Automatic testing
5. Explain Tester's workbench concept with suitable diagram
6. Explain the 11 steps testing process in details.

❁ ❁ ❁

NOTES

Chapter 7

Different types of Testing

7.0 OBJECTIVES

Dear Friends,

After studying this Chapter you will be able to discuss following -

- Installation Testing
- Usability testing
- Regression testing
- Performance Testing
- Load Testing
- Stress testing
- Security testing

7.1 INTRODUCTION

There are a number of testing types installation testing is one of it in which we check the correctness of implementation procedures. In usability testing we adjust the design to suit user's needs. In regression testing to test the program again when changes are made to the

program. In performance testing we check whether some aspect of a system performs under a particular workload. Load and stress are types of performance testing. In Load testing we check performance by multiple users accessing the program concurrently and in stress testing we check whether the system can sustain in minimum resources. Security testing tries to confirm whether the system is capable of protection from unauthorized access.

7.2 INSTALLATION TESTING

Installation testing (Implementation testing) is a kind of quality assurance work in the software industry that focuses on what customers will need to do to install and set up the new software successfully. The testing process may involve full, partial or upgrades install/uninstall processes.

This testing is typically done by the software testing engineer in conjunction with the configuration manager.

Implementation testing is usually defined as testing which places a compiled version of code into the testing or pre-production environment, from which it may or may not progress into production. This generally takes place outside of the software development environment to limit code corruption from other future releases which may reside on the development network.

The simplest installation approach is to run an install program, sometimes called package software. This package software typically uses a setup program which acts as a multi-configuration wrapper and which may allow the software to be installed on a variety of machine and/or operating environments. Every possible configuration should receive an appropriate level of testing so that it can be released to customers with confidence.

In distributed systems, particularly where software is to be released into an already live target environment (such as an operational website) installation (or software deployment as it is sometimes called) can involve database schema changes as well as the installation of new software. Deployment plans in such circumstances may include back-out procedures whose use is intended to roll the target environment back if the deployment is unsuccessful. Ideally, the deployment plan itself should be tested in an environment that is a replica of the live environment. A factor that can increase the organizational requirements of such an exercise is the need to synchronize the data in the test deployment environment with that in the live environment with minimum disruption to live operation. This type of implementation may include testing of the processes which take place during the installation or upgrade of a multi-tier application. This type of testing is commonly compared to a dress rehearsal or may even be called a “dry run”.

7.3 USABILITY TESTING

Usability testing is the process of observing users' reactions to a product and adjusting the design to suit their needs. Marketing knows usability testing as "focus groups" and while the two differ in intent many of the principles and processes are the same.

In usability testing a basic model or prototype of the product is put in front of evaluators who are representative of typical end-users. They are then set a number of standard tasks which they must complete using the product. Any difficulty or obstructions they encounter are then noted by a host or observers and design changes are made to the product to correct these. The process is then repeated with the new design to evaluate those changes.

There are some fairly important tenets of usability testing that must be understood:

- ***Users are not testers, engineers or designers –*** you are not asking the users to make design decisions about the software. Users will not have a sufficiently broad technical knowledge to make decisions which are right for everyone. However, by seeking their opinion the development team can select the best of several solutions.
- ***You are testing the product and not the users –*** all too often developers believe that it's a 'user' problem when there is trouble with an interface or design element. Users should be able to 'learn' how to use the software if they are taught properly! Maybe if the software is designed properly, they won't have to learn it at all?
- ***Selection of end-user evaluators is critical –*** You must select evaluators who are directly representative of your end-users. Don't pick just anyone off the street, don't use management and don't use technical people unless they are your target audience.
- ***Usability testing is a design tool –*** Usability testing should be conducted early in the lifecycle when it is easy to implement changes that are suggested by the testing. Leaving it till later will mean changes will be difficult to implement. One misconception about usability studies is that a large number of evaluators is required to undertake a study. Research has shown that no more than four or five evaluators might be required. Beyond that number the amount of new information discovered diminishes rapidly and each extra evaluator offers little or nothing new and five is often convincing enough.

If all five evaluators have the same problem with the software, is it likely the problem lies with them or with the software? With one or two evaluators it could be put down to personal quirks.

With five it is beyond a shadow of a doubt.

The proper way to select evaluators is to profile a typical end-user and then solicit the services of individuals who closely fit that profile. A profile should consist of factors such as age, experience, gender, education, prior training and technical expertise.

7.4 REGRESSION TESTING

Regression testing is any type of software testing that seeks to uncover software errors after changes to the program (e.g. bugfixes or new functionality) have been made, by retesting the program. The intent of regression testing is to assure that a change, such as a bugfix, did not introduce new bugs. Regression testing can be used to test the system efficiently by systematically selecting the appropriate minimum suite of tests needed to adequately cover the affected change. *Common method*s of regression testing include rerunning previously run tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged. "One of the main reasons for regression testing is that it's often extremely difficult for a programmer to figure out how a change in one part of the software will echo in other parts of the software." This is done by comparing results of previous tests to results of the current tests being run.

Regression Testing attempts to verify:

- That the application works as specified even after the changes/additions/modification were made to it. The original functionality continues to work as specified even after changes/additions/modification to the software application.
- The changes/additions/modification to the software application have not introduced any new bugs.

When is Regression Testing necessary?

Regression Testing plays an important role in any Scenario where a change has been made to a previously tested software code. Regression Testing is hence an important aspect in various Software Methodologies where software changes enhancements occur frequently.

Any Software Development Project is invariably faced with requests for changing Design, code, features or all of them.

Some Development Methodologies embrace change.

For example ‘Extreme Programming’ Methodology advocates applying small incremental changes to the system based on the end user feedback.

Each change implies more Regression Testing needs to be done to ensure that the System meets the Project Goals.

Why is Regression Testing important?

Any Software change can cause existing functionality to break.

Changes to a Software component could impact dependent Components.

It is commonly observed that a Software fix could cause other bugs.

All this affects the quality and reliability of the system. Hence Regression Testing, since it aims to verify all this, is very important.

7.2 to 7.4 Check your Progress

Fill in the blanks

1. _____ testing is the process of observing users' reactions to a product and adjusting the design to suit their needs.
2. The intent of _____ testing is to assure that a change, such as a bug fix, did not introduce new bugs.
3. Installation testing is also known as _____ testing.

7.5 PERFORMANCE TESTING

Performance testing is testing that is performed, to determine how fast some aspect of a system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system, such as scalability, reliability and resource usage.

Performance testing is a subset of Performance engineering, an emerging computer science practice which strives to build performance into the design and architecture of a system, prior to the onset of actual coding effort.

Performance testing can serve different purposes.

- It can demonstrate that the system meets performance criteria.
- It can compare two systems to find which performs better.
- Or it can measure what parts of the system or workload causes the system to perform badly.

Many performance tests are undertaken without due consideration to the setting of realistic performance goals. The first question from a business perspective should always be "why are we performance testing?". These considerations are part of the business case of the testing. Performance goals will differ depending on the application technology and purpose.

Performance testing can be performed across the web, and even done in different parts of the country, since it is known that the response times of the internet itself vary regionally. It can also be done in-house, although routers would then need to be configured to introduce the lag

what would typically occur on public networks. Loads should be introduced to the system from realistic points. For example, if 50% of a system's user base will be accessing the system via a 56K modem connection and the other half over a T1, then the load injectors (computers that simulate real users) should either inject load over the same connections (ideal) or simulate the network latency of such connections, following the same user profile.

It is always helpful to have a statement of the likely peak numbers of users that might be expected to use the system at peak times. If there can also be a statement of what constitutes the maximum allowable 95 percentile response time, then an injector configuration could be used to test whether the proposed system met that specification.

Performance testing is further divided into 1. Load testing and , 2. Stress testing.

7.5.1 Load Testing

The term *load testing* is used in different ways in the professional software testing community. *Load testing* generally refers to the practice of modeling the expected usage of a software program by simulating multiple users accessing the program concurrently. As such, this testing is most relevant for multi-user systems; often one built using a client/server model, such as web servers. However, other types of software systems can also be load tested. For example, a word processor or graphics editor can be forced to read an extremely large document; or a financial package can be forced to generate a report based on several years' worth of data. The most accurate load testing simulates actual use, as opposed to testing using theoretical or analytical modeling.

Load and performance testing analyzes software intended for a multi-user audience by subjecting the software to different amounts of virtual and live users while monitoring performance measurements under these different loads. Load and performance testing is usually conducted in a test environment identical to the production environment before the software system is permitted to go live.

As an example, a web site with shopping cart capability is required to support 100 concurrent users broken out into following activities:

- 25 Virtual Users (VUsers) log in, browse through items and then log off
- 25 VUsers log in, add items to their shopping cart, check out and then log off
- 25 VUsers log in, return items previously purchased and then log off
- 25 VUsers just log in without any subsequent activity

A test analyst can use various load testing tools to create these VUsers and their activities. Once the test has started and reached a steady state, the application is being tested at the100 VUser load as described above. The application's performance can then be monitored and captured.

The specifics of a load test plan or script will generally vary across organizations. For example, in the bulleted list above, the first item could represent 25 VUsers browsing unique items, random items, or a selected set of items depending upon the test plan or script developed. However, all load test plans attempt to simulate system performance across a range of anticipated peak workflows and volumes. The criteria for passing or failing a load test (pass/fail criteria) are generally different across organizations as well. There are no standards specifying acceptable load testing performance metrics.

A common misconception is that load testing software provides record and playback capabilities like regression testing tools. Load testing tools analyze the entire OSI protocol stack whereas most regression testing tools focus on GUI performance. For example, a regression testing tool will record and playback a mouse click on a button on a web browser, but a load testing tool will send out hypertext the web browser sends after the user clicks the button. In a multiple-user environment, load testing tools can send out hypertext for multiple users with each user having a unique login ID, password, etc.

The popular load testing tools available also provide insight into the causes for slow performance. There are numerous possible causes for slow system performance, including, but not limited to, the following:

- Application server(s) or software
- Database server(s)
- Network – latency, congestion, etc.
- Client-side processing
- Load balancing between multiple servers

Load testing is especially important if the application, system or service will be subject to a service level agreement or SLA.

7.5.2 Stress testing

Stress testing is a form of testing that is used to determine the stability of a given system or entity. It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. Stress testing may have a more specific meaning in certain industries, such as fatigue testing for materials.

In software testing, a system stress test refers to tests that put a greater emphasis on robustness, availability, and error handling under a

heavy load, rather than on what would be considered correct behavior under normal circumstances. In particular, the goals of such tests may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space), unusually high concurrency, or denial of service attacks.

Examples:

- A web server may be stress tested using scripts, bots, and various denial of service tools to observe the performance of a web site during peak loads.

Stress testing may be contrasted with load testing:

- Load testing examines the entire environment and database, while measuring the response time, whereas stress testing focuses on identified transactions, pushing to a level so as to break transactions or systems.
- During stress testing, if transactions are selectively stressed, the database may not experience much load, but the transactions are heavily stressed. On the other hand, during load testing the database experiences a heavy load, while some transactions may not be stressed.
- System stress testing, also known as stress testing, is loading the concurrent users over and beyond the level that the system can handle, so it breaks at the weakest link within the entire system.

  Stress testing defines a scenario and uses a specific algorithm to determine the expected impact on a portfolio's return should such a scenario occur. There are three types of scenarios:

- Extreme event: hypothesize the portfolio's return given the recurrence of a historical event. Current positions and risk exposures are combined with the historical factor returns.
- Risk factor shock: shock any factor in the chosen risk model by a user-specified amount. The factor exposures remain unchanged, while the covariance matrix is used to adjust the factor returns based on their correlation with the shocked factor.
- External factor shock: instead of a risk factor, shock any index, macro-economic series (e.g., oil prices), or custom series (e.g., exchange rates). Using regression analysis, new factor returns are estimated as a result of the shock.

  In an exponentially weighted stress test, historical periods more like the defined scenario receive a more significant weighting in the predicted outcome. The defined decay rate lets the tester manipulate the relative importance of the most similar periods. In the standard stress test, each period is equally weighted.

7.6 SECURITY TESTING

Security testing is a process to determine that an information system protects data and maintains functionality as intended.

The six basic security concepts that need to be covered by security testing are: confidentiality, integrity, authentication, availability, authorization and non-repudiation. Security testing as a term has a number of different meanings and can be completed in a number of different ways. As such a Security Taxonomy helps us to understand these different approaches and meanings by providing a base level to work from.

Security measures

- A security measure which protects against the disclosure of information to parties other than the intended recipient that is by no means the only way of ensuring the security.

Integrity

- A measure intended to allow the receiver to determine that the information which it is providing is correct.
- Integrity schemes often use some of the same underlying technologies as confidentiality schemes, but they usually involve adding additional information to a communication to form the basis of an algorithmic check rather than the encoding all of the communication.

Authentication

- It is a type of security testing in which one will enter different combinations of usernames and passwords and will check whether only the authorized people are able to access it or not.
- The process of establishing the identity of the user.
- Authentication can take many forms including but not limited to: passwords, biometrics, radio frequency identification, etc.

Authorization

- The process of determining that a requester is allowed to receive a service or perform an operation.
- Access control is an example of authorization.

Availability

- Assuring information and communications services will be ready for use when expected.
- Information must be kept available to authorized persons when they need it.

Non-repudiation

- A measure intended to prevent the later denial that an action happened, or a communication that took place etc.
- In communication terms this often involves the interchange of authentication information combined with some form of provable time stamp.

Confidentiality

Security Testing Taxonomy

Common terms used for the delivery of security testing;

- **Discovery -** The purpose of this stage is to identify systems within scope and the services in use. It is not intended to discover vulnerabilities, but version detection may highlight deprecated versions of software / firmware and thus indicate potential vulnerabilities.
- **Vulnerability Scan -** Following the discovery stage this looks for known security issues by using automated tools to match conditions with known vulnerabilities. The reported risk level is set automatically by the tool with no manual verification or interpretation by the test vendor. This can be supplemented with credential based scanning that looks to remove some common false positives by using supplied credentials to authenticate with a service (such as local windows accounts).
- **Vulnerability Assessment -** This uses discovery and vulnerability scanning to identify security vulnerabilities and places the findings into the context of the environment under test. An example would be removing common false positives from the report and deciding risk levels that should be applied to each report finding to improve business understanding and context.
- **Security Assessment -** Builds upon Vulnerability Assessment by adding manual verification to confirm exposure, but does not include the exploitation of vulnerabilities to gain further access. Verification could be in the form of authorized access to a system to confirm system settings and involve examining logs, system responses, error messages, codes, etc. A Security Assessment is looking to gain a broad coverage of the systems under test but not the depth of exposure that a specific vulnerability could lead to.
- **Penetration Test -** Penetration test simulates an attack by a malicious party. Building on the previous stages and involves exploitation of found vulnerabilities to gain further access. Using this approach will result in an understanding of the ability of an

attacker to gain access to confidential information, affect data integrity or availability of a service and the respective impact. Each test is approached using a consistent and complete methodology in a way that allows the tester to use their problem solving abilities, the output from a range of tools and their own knowledge of networking and systems to find vulnerabilities that would/ could not be identified by automated tools. This approach looks at the depth of attack as compared to the Security Assessment approach that looks at the broader coverage.

- **Security Audit -** Driven by an Audit / Risk function to look at a specific control or compliance issue. Characterized by a narrow scope, this type of engagement could make use of any of the earlier approaches discussed (vulnerability assessment, security assessment, penetration test).

1. **Security Review** - Verification that industry or internal security standards have been applied to system components or product. This is typically completed through gap analysis and utilises build /code reviews or by reviewing design documents and architecture diagrams. This activity does not utilise any of the earlier approaches (Vulnerability Assessment, Security Assessment, Penetration Test, Security Audit).

7.5 & 7.6 Check your Progress

Q1. Fill in the blanks

1. In _____ testing multiple users access the program concurrently.
2. _____testing is a process to determine that an information system protects data and maintains functionality as intended.

Q2. State whether True or False

1. Stress testing involves testing in normal operational capacity.
2. Performance testing validates and verifies resource usage.

7.7 SUMMARY

Installation testing focuses on what customers will need to do to install and set up the new software successfully. In usability testing user's reactions to a product are observed design is adjusted to suit their needs. The intent of regression testing is to assure that a change, such as a bugfix, did not introduce new bugs. Regression Testing is an important

aspect in various Software Methodologies where software changes enhancements occur frequently. Performance testing is used to validate and verify quality attributes of the system, such as scalability, reliability and resource usage.

Load testing refers to the practice of modeling the expected usage of a software program by simulating multiple users accessing the program concurrently. Stress testing involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. Security testing is a process to determine that an information system protects data and maintains functionality as intended.

7.8 CHECK YOUR PROGRESS- ANSWERS

7.1 to 7.3

Q1. Fill in the blanks

1. Usability
2. Regression
3. Implementation

7.5 to 7.6

Q1.

1. Load
2. Security

Q2.

1. False
2. True

7.9 QUESTIONS FOR SELF – STUDY

1. Write a note on a) Installation testing b) Usability testing
2. What do you mean by regression testing? Explain its necessity and importance.
3. Describe Performance testing. Also explain Load and Stress testing.
4. What is security testing? Explain security testing measures.

❁ ❁ ❁

NOTES

NOTES

CHAPTER 8

Static & Dynamic Testing

8.0 OBJECTIVES

Dear Friends,

After studying this chapter you will be able to -

- Describe Static Testing Techniques
- Discuss Review and types: Informal Review, Technical or peer review, Walkthrough, Inspection
- Explain Static analysis
- Describe Data flow analysis
- Describe Control flow analysis
- Describe Cyclomatic complexity Analysis
- Explain Dynamic testing

8.1 INTRODUCTION

Static testing techniques the software is checked without executing

it i.e. checking its design and documentation. Here the probable defects are detected earlier so it saves a lot of money in case defects found later. The two bread categories of static testing are Reviews and Static analysis. In Data-flow analysis information is gathered about the possible set of values calculated at various points in a computer program. In control flow analysis a control flow graph is used to show how events in the program are sequenced. Cyclomatic complexity analysis is a way of measuring complexity.

8.2 STATIC TESTING TECHNIQUES

Static testing techniques do not execute the software that is being tested. The main manual activity is to examine a work product and make comments about it.

Static testing techniques are categorised into manual (reviews) or automated (static analysis).

8.2.1 Reviews

Defects detected during reviews early in the life cycle are often much cheaper to remove than those detected while running tests (e.g. defects found in requirements).

- Any software work product can be reviewed, including requirement specifications, design specifications, code, test plans, test cases, test scripts, user guides or web pages.
- Reviews are a way of testing software work products (including code) and can be performed well before dynamic test execution

Benefits of Reviews

- Early defect detection and correction,
- Development productivity improvements,
- Reduced development timescales
- Reduced testing cost and time,
- Lifetime cost reductions,
- Fewer defects at later stage and improved communication

Typical defects that are easier to find in reviews are:

- Deviations from standards
- Requirement defects
- Design defects
- Incorrect interface specifications etc

A typical formal review has the following main phases:

- Planning: selecting the personnel, allocating roles; defining the entry and exit criteria for more formal review types (e.g. inspection); and selecting which parts of documents to look at.
- Kick-off: distributing documents; explaining the objectives, process and documents to the participants; and checking entry criteria (for more formal review types).
- Individual preparation: work done by each of the participants on their own before the review meeting, noting potential defects, questions and comments.
- Review meeting: discussion or logging, with documented results or minutes (for more formal review types). The meeting participants may simply note defects, make recommendations for handling the defects, or make decisions about the defects.
- Rework: fixing defects found, typically done by the author.
- Follow-up: checking that defects have been addressed, gathering metrics and checking on exit criteria (for more formal review types).

A typical formal review will include the roles below:

- Manager: decides on the execution of reviews, allocates time in project schedules and determines if the review objectives have been met.
- Moderator: the person who leads the review of the document or set of documents, including planning the review, running the meeting, and follow-up after the meeting. If necessary, the moderator may mediate between the various points of view and is often the person upon whom the success of the review rests.
- Author: the writer or person with chief responsibility for the document(s) to be reviewed.
- Reviewers: individuals with a specific technical or business background (also called checkers or inspectors) who, after the necessary preparation, identify and describe findings (e.g. defects) in the product under review. Reviewers should be chosen to represent different perspectives and roles in the review process and they take part in any review meetings.
- Scribe (or recorder): documents all the issues, problems and open points that were identified during the meeting

Review Checklist

Compliance with standards – Does the requirements specification comply with ISD or tailored Branch/project-level standards and naming conventions?

Completeness of Specifications – Does the requirements specification

document address all known requirements? Have 'TBD' requirements been kept to a minimum, or eliminated entirely?

Clarity – Are the requirements clear enough to be turned over to an independent group for implementation?

Consistency – Are the specifications consistent in notation, terminology, and level of functionality? Are any required algorithms mutually compatible?

External Interfaces – Have external interfaces been adequately defined?

Testability – Are the requirements testable? Will the testers be able to determine whether each requirement has been satisfied?

Design-Neutrality – Does the requirements specification state what actions are to be performed, rather than how these actions will be performed?

Readability – Does the requirements specification use the language of the intended testers and users of the system, not software jargon?

Level of Detail – Are the requirements at a fairly consistent level of detail? Should any particular requirement be specified in more detail? In less detail?

Requirements Singularity – Does each requirement address a single concept, topic, element, or value?

Definition of Inputs and Outputs – Have the internal interfaces, i.e., the required inputs to and outputs from the software system, been fully defined? Have the required data transformations been adequately specified?

Scope – Does the requirements specification adequately define boundaries for the scope of the target software system? Are any essential requirements missing?

Design Constraints – Are all stated design and performance constraints realistic and justifiable?

Traceability – Has a bidirectional traceability matrix been provided?

Review Types

- Informal Reviews
- Walkthroughs
- Formal technical Reviews
- Inspections

8.2.1.1 Informal review

The Key characteristics of informal review are as follows:

- No formal process
- There may be pair programming or a technical lead reviewing

designs and code (led by individual)

- Optionally may be documented or undocumented
- May vary in usefulness depending on the reviewer
- Main purpose: inexpensive way to get some benefit

8.2.1.2 Walkthroughs

The Key characteristics of Walkthroughs are as follows:

- Meeting led by author
- Scenarios, dry runs, peer group
- Open-ended sessions
- Optionally a pre-meeting preparation of reviewers, review report, list of findings and scribe(who is not the author)
- May vary in practice from quite informal to very formal;
- Main purposes: learning, gaining understanding, defect finding.

8.2.1.3 Formal technical Reviews

The Key characteristics of Formal technical Reviews are as follows:

- Documented, defined defect-detection process that includes peers and technical experts
- May be performed as a peer review without management participation
- Ideally led by trained moderator (not the author)
- Pre-meeting preparation
- Optionally the use of checklists, review report, list of findings and management participation
- May vary in practice from quite informal to very formal
- Main purposes: discuss, make decisions, evaluate alternatives, find defects, solve technical problems and check conformance to specifications and standards

8.2.1.4 Inspection

The Key characteristics of Inspection are as follows:

- led by trained moderator (not the author)
- Usually peer examination
- Defined roles
- Includes metrics

- Formal process based on rules and checklists with entry and exit criteria
- Pre-meeting preparation
- Inspection report, list of findings
- Formal follow-up process
- Optionally, process improvement and reader
- Main purpose: find defects

8.2.2 Static analysis

Static analysis, also called static code analysis, is a method of computer program debugging that is done by examining the code without executing the program. The process provides an understanding of the code structure, and can help to ensure that the code adheres to industry standards. Automated tools can assist programmers and developers in carrying out static analysis. The process of scrutinizing code by visual inspection alone (by looking at a printout, for example), without the assistance of automated tools, is sometimes called program understanding or program comprehension.

The principal advantage of static analysis is the fact that it can reveal errors that do not manifest themselves until a disaster occurs weeks, months or years after release. Nevertheless, static analysis is only a first step in a comprehensive software quality-control regime. After static analysis has been done, dynamic analysis is often performed in an effort to uncover subtle defects or vulnerabilities. In computer terminology, static means fixed, while dynamic means capable of action and/or change. Dynamic analysis involves the testing and evaluation of a program based on execution. Static and dynamic analyses, considered together, are sometimes referred to as glass-box testing.

The value of static analysis is:

- Early detection of defects prior to test execution
- Early warning about suspicious aspects of the code or design, by the calculation of metrics, such as a high complexity measure.
- Identification of defects not easily found by dynamic testing
- Detecting dependencies and inconsistencies in software models
- Improved maintainability of code and design
- Prevention of defects

Static Analysis by tools

The objective is to find defects in software source code and software models.

Static analysis is performed without actually executing the software being examined by the tool Static analysis can locate defects that are hard to find in testing.

As with reviews, static analysis finds defects rather than failures.

Static analysis tools analyze program code (e.g. control flow and data flow), as well as generated output such as HTML and XML

Typical defects discovered by static analysis tools include:

- Referencing a variable with an undefined value
- Inconsistent interface between modules and components
- Variables that are never used
- Unreachable (dead) code
- Programming standards violations
- Security vulnerabilities
- Syntax violations of code and software models.

8.2 Check your Progress

Q1. Fill in the blank.

1. _____ led by trained moderator
2. The main manual activity in _____ testing is to examine a work product and make comments about it.

Q2. True or False.

1. In walkthroughs Scenarios, dry runs, peer group are used.
2. Static analysis is done by examining the code with executing the program.

8.3 DATA FLOW ANALYSIS

Data-flow analysis is a technique for gathering information about the possible set of values calculated at various points in a computer program. A program's control flow graph (CFG) is used to determine those parts of a program to which a particular value assigned to a variable might propagate. The information gathered is often used by compilers when optimizing a program. A canonical example of a data-flow analysis is reaching definitions.

A simple way to perform dataflow analysis of programs is to set up dataflow equations for each node of the control flow graph and solve them by repeatedly calculating the output from the input locally at each node until the whole system stabilizes, i.e., it reaches a fixpoint. This general approach was developed by Gary Kildall while teaching at the Naval Postgraduate School.

It is the process of collecting information about the way the variables are used, defined in the program. Data flow analysis attempts to obtain particular information at each point in a procedure. Usually, it is enough to obtain this information at the boundaries of basic blocks, since from that it is easy to compute the information at points in the basic block. In forward flow analysis, the exit state of a block is a function of the block's entry state. This function is the composition of the effects of the statements in the block. The entry state of a block is a function of the exit states of its predecessors. This yields a set of data flow equations:

For each block b:

$out_b = trans_b(in_b)$

$$in_b = join_{p \in pred_b}(out_p)$$

In this, $trans_b$ is the transfer function of the block *b*. It works on the entry state in_b, yielding the exit state out_b. The join operation *join* combines the exit states of the predecessors $p \in pred_b$ of *b*, yielding the entry state of *b*.

After solving this set of equations, the entry and / or exit states of the blocks can be used to derive properties of the program at the block boundaries. The transfer function of each statement separately can be applied to get information at a point inside a basic block.

Each particular type of data flow analysis has its own specific transfer function and joins operation. Some data flow problems require backward flow analysis. This follows the same plan, except that the transfer function is applied to the exit state yielding the entry state, and the join operation works on the entry states of the successors to yield the exit state.

The entry point (in forward flow) plays an important role: Since it has no predecessors, its entry state is well defined at the start of the analysis. For instance, the set of local variables with known values is empty. If the control flow graph does not contain cycles (there were no explicit or implicit loops in the procedure) solving the equations is straightforward. The control flow graph can then be topologically sorted; running in the order of this sort, the entry states can be computed at the start of each block, since all predecessors of that block have already been processed, so their exit states are available. If the control flow graph does contain cycles, a more advanced algorithm is required.

8.4 CONTROL FLOW ANALYSIS

The DMS Software Reengineering Toolkit provides support for computing various kinds of control and data flows. Serious program analysis and transformation tasks often require a deep understanding of information flows that occur between program components. These

information flows are induced by the various runtime entity declarations (variables and procedures) and the references, explicit or implied, between them, which are in turn implied by the semantics of the language being processed. Since these semantics are not always directly visible in the syntax of the language, trivial syntax analysis cannot easily discover them, and typically an analysis procedure that understands the semantic implications of the syntax needs to be constructed to make the information flows accessible.

The process of making such information flows visible is called "flow analysis". The result of such an analysis is usually a graph, often with annotations. Commonly, a control flow graph ("flowchart") is produced, and data flow analyses augment that graph with additional arcs or annotations on the nodes of the control flow graph ("facts").

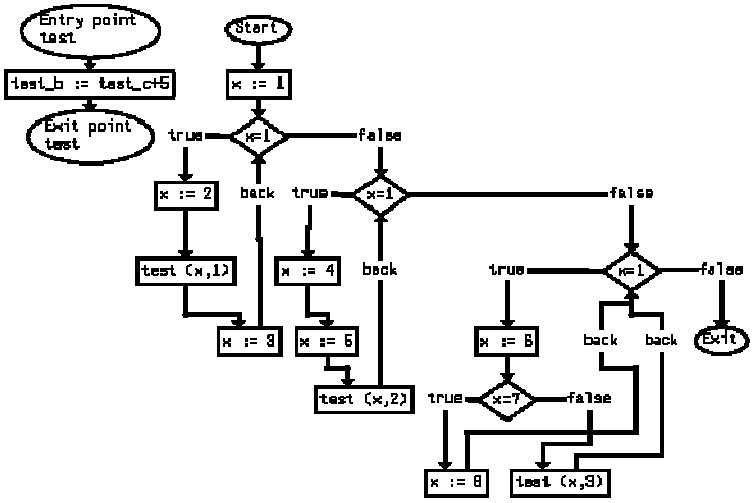
Control Flow Analysis and Graphs

A control flow graph shows how events in the program are sequenced. Traditional flowcharts are limited to executable statements ("boxes"), conditional tests ("diamonds") and subroutine calls ("hexagons"). DMS flow graphs represent these concepts directly as nodes, and attach the AST for the originating tree to each control flow node. Flow graphs for modern languages require more sophistication to represent more complex control flows. DMS flow graphs can include regions (e.g., for exceptions), indirect GOTOs, fork-join and partial-order nodes to represent parallelism (including concurrency or simply lack of defined order). As examples, DMS-derived C control flow graphs contain fork-join nodes used to model the C standard's notion of "sequence points", where sub-expression with unspecified order must be complete before further computation occurs (e.g., the arguments to a function are evaluated in an arbitrary order, but the function call cannot occur until all are evaluated). Control flow graphs for GCC C source code can also contain indirect GOTO nodes, to model GCC's "goto *exp;" construct.

Control flow graphs for a specific language (e.g, C or a dialect) is often implemented in DMS by using an attribute grammar evaluator that propagates control points around the abstract syntax tree and assembles those control points into a completed graph. These evaluators use a DMS Control Flow graph domain and a supporting library of control flow graph facilities that it provides. Using these evaluators and this library, it is generally straightforward to construct flow graphs for many modern languages.

Often it is important to know when one computation always occurs before another ("dominates"). The library can compute dominators and post-dominators from standard DMS control flow graphs. It is also useful to know under what condition some control flow node executes; the library also computes this and decorates the control flow graph with such "control dependence" arcs.

Example of control flow graph



8.5 CYCLOMATIC COMPLEXITY ANALYSIS

Cyclomatic Code Complexity was first introduced by Thomas McCabe in 1976. In 1976, Thomas McCabe published a paper arguing that code complexity is defined by its control flow. Since that time, others have identified different ways of measuring complexity (e.g. data complexity, module complexity, algorithmic complexity, call-to, call-by, etc.). Although these other methods are effective in the right context, it seems to be generally accepted that control flow is one of the most useful measurements of complexity, and high complexity scores have been shown to be a strong indicator of low reliability and frequent errors.

Overview

This measure provides a single ordinal number that can be compared to the complexity of other programs. It is one of the most widely accepted static software metrics and is intended to be independent of language and language format.

Code Complexity is a measure of the number of linearly-independent paths through a program module and is calculated by counting the number of decision points found in the code (if, else, do, while, throw, catch, return, break etc.).

Technical Specification

Cyclomatic Complexity for a software module calculated based on graph theory is based on the following equation:

⊟Collapse

CC=E-N+p

Where

- CC = Cyclomatic Complexity
- E = the number of edges of the graph
- N = the number of nodes of the graph
- p = the number of connected components

Further academic information on the specifics of this can be found here.

From a laymans perspective the above equation can be pretty daunting to comprehend. Fortunately there is a simpler equation which is easier to understand and implement by following the guidelines shown below:

- Start with 1 for a straight path through the routine.
- Add 1 for each of the following keywords or their equivalent: if, while, repeat, for, and, or.
- Add 1 for each case in a switch statement.

8.6 DYNAMIC TESTING

Dynamic testing (or dynamic analysis) is a term used in software engineering to describe the testing of the dynamic behavior of code. That is, dynamic analysis refers to the examination of the physical response from the system to variables that are not constant and change with time. In dynamic testing the software must actually be compiled and run; Actually Dynamic Testing involves working with the software, giving input values and checking if the output is as expected. These are the Validation activities. Unit Tests, Integration Tests, System Tests and Acceptance Tests are few of the Dynamic Testing methodologies. Dynamic testing means testing based on specific test cases by execution of the test object or running programs. Dynamic testing is used to test software through executing it.

8.3 to 8.6 Check your Progress

Fill in the blanks

1. In _____testing the software must actually be compiled and run.
2. _____ is a technique for gathering information about the possible set of values calculated at various points in a computer program.
3. A _____ shows how events in the program are sequenced.

8.7 SUMMARY

Static testing techniques are categorized into reviews or static analysis. Reviews are a way of testing software work products (including code) and can be performed well before dynamic test execution.

Informal Reviews, Walkthroughs, Formal technical Reviews, Inspections are the types of Reviews. Static analysis is a method of computer program debugging that is done by examining the code without executing the program. The principal advantage of static analysis is the fact that it can reveal errors that do not manifest themselves until a disaster occurs weeks, months or years after release.

Data-flow analysis is a technique for gathering information about the possible set of values calculated at various points in a computer program. A program's control flow graph (CFG) is used to determine those parts of a program to which a particular value assigned to a variable might propagate. A control flow graph shows how events in the program are sequenced. Code Complexity is a measure of the number of linearly-independent paths through a program module and is calculated by counting the number of decision points found in the code (if, else, do, while, throw, catch, return, break etc.). Dynamic analysis refers to the examination of the physical response from the system to variables that are not constant and change with time.

8.8 CHECK YOUR PROGRESS – ANSWERS

8.2
Q1.
1. Inspection
2. Static

Q2.
1. True
2. False

8.3 to 8.6
Q1.
1. Dynamic
2. Data-flow analysis
3. Control flow graph

8.9 QUESTIONS FOR SELF – STUDY

1. Explain in details about software reviews.
2. Describe the main phases in reviews.
3. Explain the various roles in reviews.
4. Write a note on

   a) Informal Reviews b) Walkthroughs c) Formal technical Reviews, d) Inspections
5. Explain in details about static analysis
6. Write short notes on a) Data flow analysis b) Control flow analysis c) Cyclometric complexity.

❁ ❁ ❁

NOTES

CHAPTER 9

Black Box & White Box Testing
(Test Case Design Techniques)

9.0 OBJECTIVES

Dear Friends,

After studying this chapter you will be able to -

- Explain Functional Testing (Black Box)
- Discuss Structural Testing (White Box)
- Discuss Domain Testing
- Describe Non functional testing techniques
- Describe Validation testing Activities
- Describe Black box vs. White Box

9.1 INTRODUCTION

Functional testing is based on requirements specifications. Equivalence partitioning, boundary value analysis and Cause effect graphics are its methods. Structural testing tries to uncover the errors by looking into the code. Domain testing checks the system related with a particular area of interest. The aim of both validation testing is to ensure that the software product is made according to the requirements of the client and does indeed fulfill the intended purpose.

9.2 FUNCTIONAL TESTING (BLACK BOX)

Functional testing is a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered. Functional testing differs from system testing in that functional testing "*verifies* a program by checking it against ... design documents or specifications", while system testing "*validates* a program by checking it against the published user or system requirements".

Functional testing typically involves five steps -

1. The identification of functions that the software is expected to perform
2. The creation of input data based on the function's specifications
3. The determination of output based on the function's specifications
4. The execution of the test case
5. The comparison of actual and expected outputs

9.2.1 Equivalence partitioning

In this method the input domain data is divided into different equivalence data classes. This method is typically used to reduce the total number of test cases to a finite set of testable test cases, still covering maximum requirements.

In short it is the process of taking all possible test cases and placing them into classes. One test value is picked from each class while testing.

E.g.: If you are testing for an input box accepting numbers from 1 to 1000 then there is no use in writing thousand test cases for all 1000 valid input numbers plus other test cases for invalid data.

Using equivalence partitioning method above test cases can be divided into three sets of input data called as classes. Each test case is a representative of respective class.

So in above example we can divide our test cases into three equivalence classes of some valid and invalid inputs.

Test cases for input box accepting numbers between 1 and 1000 using Equivalence Partitioning:

1) One input data class with all valid inputs. Pick a single value from range 1 to 1000 as a valid test case. If you select other values between 1 and 1000 then result is going to be same. So one test case for valid input data should be sufficient.

2) Input data class with all values below lower limit. I.e. any value below 1, as a invalid input data test case.

3) Input data with any value greater than 1000 to represent third invalid input class.

So using equivalence partitioning you have categorized all possible test cases into three classes. Test cases with other values from any class should give you the same result.

We have selected one representative from every input class to design our test cases. Test case values are selected in such a way that largest number of attributes of equivalence class can be exercised.

Equivalence partitioning uses fewest test cases to cover maximum requirements.

9.2.2 Boundary Value Analysis

It's widely recognized that input values at the extreme ends of input domain cause more errors in system. More application errors occur at the boundaries of input domain. 'Boundary value analysis' testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

Test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis:

1) Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.

2) Test data with values just below the extreme edges of input domains i.e. values 0 and 999.

3) Test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

Boundary value analysis is often called as a part of stress and negative testing.

Note: There is no hard-and-fast rule to test only one value from each equivalence class you created for input domains. You can select multiple valid and invalid values from each equivalence class according to your needs and previous judgments.

E.g. if you divided 1 to 1000 input values in valid data equivalence class, then you can select test case values like: 1, 11, 100, 950 etc. Same case for other test cases having invalid data classes.

This should be a very basic and simple example to understand the Boundary value analysis and Equivalence partitioning concept.

9.2.3 Cause- Effect graphing

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

Constraints may be added to the causes and effects. These are represented as edges labelled with the constraint symbol using a dashed line. For causes, valid constraint symbols are E (exclusive), O (one and only one), and I (at least one). The exclusive constraint states that both causes1 and cause2 cannot be true simultaneously. The Inclusive (at least one) constraint states that at least one of the causes 1, 2 or 3 must be true. The OaOO (One and Only One) constraint states that only one of the causes 1, 2 or 3 can be true.

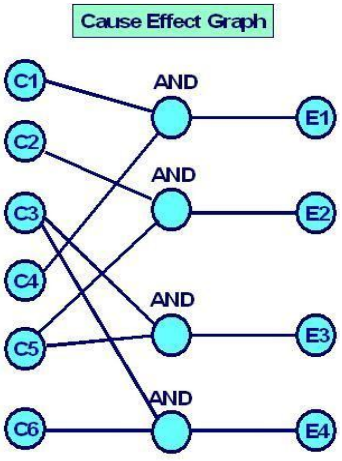
For effects, valid constraint symbols are R (Requires) and M (Mask). The Requires constraint states that if cause 1 is true, then cause 2 must be true, and it is impossible for 1 to be true and 2 to be false. The mask constraint states that if effect 1 is true then effect 2 is false. (Note that the mask constraint relates to the effects and not the causes like the other constraints.

The graph's direction is as follows:

Causes --> intermediate nodes --> Effects

The graph can always be rearranged so there is only one node between any input and any output. They are conjunctive normal form and disjunctive normal form.

A cause-effect graph is useful for generating a reduced decision table.



9.2.4 Syntax testing

Syntax Testing uses a model of the formally-defined syntax of the inputs to a component.

The syntax is represented as a number of rules each of which defines the possible means of production of a symbol in terms of sequences of, iterations of, or selections between other symbols

Test cases with valid and invalid syntax are designed from the formally defined syntax of the inputs to the component.

Test cases with valid syntax shall be designed to execute options which are derived from rules which shall include those that follow, although additional rules may also be applied where appropriate:

- Whenever a selection is used, an option is derived for each alternative by replacing the selection with that alternative;
- Whenever iteration is used, at least two options are derived, one with the minimum number of iterated symbols and the other with more than the minimum number of repetitions.

A test case may exercise any number of options. For each test case the following shall be identified:

- the input(s) to the component;
- option(s) exercised;
- the expected outcome of the test case.

Test cases with invalid syntax shall be designed as follows:

- a checklist of generic mutations shall be documented which can be applied to rules or parts of rules in order to generate a part of the input which is invalid;
- this checklist shall be applied to the syntax to identify specific mutations of the valid input, each of which employs at least one generic mutation;
- test cases shall be designed to execute specific mutations.

For each test case the following shall be identified:

- the input(s) to the component;
- the generic mutation(s) used;
- the syntax element(s) to which the mutation or mutations are applied;
- the expected outcome of the test case.

9.3 STRUCTURAL TESTING (WHITE BOX)

Structural testing is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality. In structural testing an internal perspective of the system, as well as programming skills, are required and used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs.

9.3.1 Coverage testing

Code coverage analysis is a structural testing technique (AKA glass box testing and white box testing). Structural testing compares test program behavior against the apparent intention of the source code. This contrasts with functional testing (AKA black-box testing), which compares test program behavior against a requirements specification. Structural testing examines how the program works, taking into account possible pitfalls in the structure and logic. Functional testing examines what the program accomplishes, without regard to how it works internally.

Structural testing is also called path testing since you choose test cases that cause paths to be taken through the structure of the program. Do not confuse path testing with the path coverage metric, explained later.

At first glance, structural testing seems unsafe. Structural testing cannot find errors of omission. However, requirements specifications sometimes do not exist, and are rarely complete. This is especially true near the end of the product development time line when the requirements specification is updated less frequently and the product itself begins to take over the role of the specification. The difference between functional and structural testing blurs near release time.

The Premise

The basic assumptions behind coverage analysis tell us about the strengths and limitations of this testing technique. Some fundamental assumptions are listed below.

- Bugs relate to control flow and you can expose Bugs by varying the control flow. For example, a programmer wrote "if (c)" rather than "if (!c)".
- You can look for failures without knowing what failures might occur and all tests are *reliable*, in that successful test runs imply program correctness. The tester understands what a correct version of the program would do and can identify differences from the correct behavior.
- Other assumptions include achievable specifications, no errors of omission, and no unreachable code.

Clearly, these assumptions do not always hold. Coverage analysis exposes some plausible bugs but does not come close to exposing all classes of bugs. Coverage analysis provides more benefit when applied to an application that makes a lot of decisions rather than data-centric applications, such as a database application.

Basic Metrics

A large variety of coverage metrics exist. This section contains a summary of some fundamental metrics and their strengths, weaknesses and issues.

The U.S. Department of Transportation Federal Aviation Administration (FAA) has formal requirements for structural coverage in the certification of safety-critical airborne systems Few other organizations have such requirements, so the FAA is influential in the definitions of these metrics.

Statement Coverage

This metric reports whether each executable statement is encountered. Declarative statements that generate executable code are considered executable statements. Control-flow statements, such as if, for, and switch are covered if the expression controlling the flow is covered as well as all the contained statements. Implicit statements, such as an omitted return, are not subject to statement coverage.

Also known as: line coverage, segment coverage C1 and basic block coverage. Basic block coverage is the same as statement coverage except the unit of code measured is each sequence of non-branching statements.

I highly discourage using the non-descriptive name C1. People sometimes incorrectly use the name C1 to identify decision coverage. Therefore this term has become ambiguous.

The chief advantage of this metric is that it can be applied directly to object code and does not require processing source code. Performance profilers commonly implement this metric.

The chief disadvantage of statement coverage is that it is insensitive to some control structures. For example, consider the following C/C++ code fragment:

```
int* p = NULL;
if (condition)
 p = &variable;
*p = 123;
```

Without a test case that causes condition to evaluate false, statement coverage rates this code fully covered. In fact, if condition ever evaluates false, this code fails. This is the most serious shortcoming of statement coverage. If-statements are very common.

Statement coverage does not report whether loops reach their termination condition - only whether the loop body was executed. With C, C++, and Java, this limitation affects loops that contain break statements.

Since do-while loops always execute at least once, statement coverage considers them the same rank as non-branching statements.

Statement coverage is completely insensitive to the logical operators (|| and &&).

Statement coverage cannot distinguish consecutive switch labels.

Test cases generally correlate more to decisions than to statements. You probably would not have 10 separate test cases for a sequence of 10 non-branching statements; you would have only one test case. For example, consider an if-else statement containing one statement in the then-clause and 99 statements in the else-clause. After exercising one of the two possible paths, statement coverage gives extreme results: either 1% or 99% coverage. Basic block coverage eliminates this problem.

One argument in favor of statement coverage over other metrics is that bugs are evenly distributed through code; therefore the percentage of executable statements covered reflects the percentage of faults discovered. However, one of our fundamental assumptions is that faults are related to control flow, not computations. Additionally, we could reasonably expect that programmers strive for a relatively constant ratio of branches to statements.
In summary, this metric is affected more by computational statements than by decisions.

Decision Coverage

This metric reports whether Boolean expressions tested in control structures (such as the if-statement and while-statement) evaluated to both true and false. The entire Boolean expression is considered one true-or-false predicate regardless of whether it contains logical-and or logical-or operators. Additionally, this metric includes coverage of switch-statement cases, exception handlers, and all points of entry and exit. Constant expressions controlling the flow are ignored.

Also known as: branch coverage, all-edges coverage, basis path coverage, C2, decision-decision-path testing. "Basis path" testing selects paths that achieve decision coverage. I discourage using the non-descriptive name C2 because of the confusion with the term C1.

The FAA makes a distinction between branch coverage and decision coverage, with branch coverage weaker than decision coverage. The FAA definition of a decision is, in part, "A Boolean expression composed of conditions and zero or more Boolean operators." So the FAA definition of decision coverage requires all Boolean expressions to evaluate to both true and false, even those that do not affect control flow. There is no precise definition of "Boolean expression." Some languages,

especially C, allow mixing integer and Boolean expressions and do not require Boolean variables be declared as Boolean. The FAA suggests using context to identify Boolean expressions, including whether expressions are used as operands to Boolean operators or tested to control flow. The suggested definition of "Boolean operator" is a built-in (not user-defined) operator with operands and result of Boolean type. The logical-not operator is exempted due to its simplicity. The C conditional operator (?:) is considered a Boolean operator if all three operands are Boolean expressions.

This metric has the advantage of simplicity without the problems of statement coverage.

A disadvantage is that this metric ignores branches within Boolean expressions which occur due to short-circuit operators. For example, consider the following C/C++/Java code fragment:

```
if (condition1 && (condition2 || function1()))
   statement1;
else
   statement2;
```

This metric could consider the control structure completely exercised without a call to function1. The test expression is true when condition1 is true and condition2 is true, and the test expression is false when condition1 is false. In this instance, the short-circuit operators preclude a call to function1.

The FAA suggests that for the purposes of measuring decision coverage, the operands of short-circuit operators (including the C conditional operator) be interpreted as decisions

Condition Coverage

Condition coverage reports the true or false outcome of each condition. A condition is an operand of a logical operator that does not contain logical operators. Condition coverage measures the conditions independently of each other.

This metric is similar to decision coverage but has better sensitivity to the control flow.

However, full condition coverage does not guarantee full decision coverage. For example, consider the following C++/Java fragment.

```
bool f(bool e) { return false; }
bool a[2] = { false, false };
if (f(a && b)) ...
if (a[int(a && b)]) ...
```

if ((a && b) ? false : false) ...

All three of the if-statements above branch false regardless of the values of a and b. However if you exercise this code with a and b having all possible combinations of values, condition coverage reports full coverage.

Path Coverage

This metric reports whether each of the possible paths in each function have been followed. A path is a unique sequence of branches from the function entry to the exit.

Also known as predicate coverage. Predicate coverage views paths as possible combinations of logical conditions

Since loops introduce an unbounded number of paths, this metric considers only a limited number of looping possibilities. A large number of variations of this metric exist to cope with loops. Boundary-interior path testing considers two possibilities for loops: zero repetitions and more than zero repetitions

For do-while loops, the two possibilities are one iteration and more than one iteration.

Path coverage has the advantage of requiring very thorough testing. Path coverage has two severe disadvantages. The first is that the number of paths is exponential to the number of branches. For example, a function containing 10 if-statements has 1024 paths to test. Adding just one more if-statement doubles the count to 2048. The second disadvantage is that many paths are impossible to exercise due to relationships of data. For example, consider the following C/C++ code fragment:

```
if (success)
   statement1;
statement2;
if (success)
   statement3;
```

Path coverage considers this fragment to contain 4 paths. In fact, only two are feasible: success=false and success=true.

Researchers have invented many variations of path coverage to deal with the large number of paths. For example, n-length sub-path coverage reports whether you exercised each path of length n branches.

9.4 DOMAIN TESTING

Domain is a specific area to which the project belongs

Domain testing It is a field of study that defines a set of common

requirements, terminology, and functionality for any software program constructed to solve a problem in that field.

The important white box testing method is domain testing. The goal is to check values taken by a variable a condition or an index and to prove that they are outside the specified or valid range. It also contains checking that the program accepts only valid input...for this domain testing us should expert on that particular domain....

Different Software Domains:

Banking

Finance

Insurance

E-Learning

Job Portal

Health Care

Shopping Portal etc..

Tester should aware and clear cut knowledge in the Domain makes him work effective in the different Domains. It is nothing but conducting testing on different domains related to various projects .It varies based on project area/Domain

9.2 to 9.4 Check your Progress

Fill in the blanks

1. In _____ test cases are selected at the edges of the equivalence classes.
2. _____ metric reports whether Boolean expressions tested in control structures evaluated to both true and false.
3. _____ is a type of black box testing that bases its test cases on the specifications of the software component under test.

9.5 NON FUNCTIONAL TESTING TECHNIQUES

Non-functional testing is the testing of a software application for its non-functional requirements. The names of many non-functional tests are often used interchangeably because of the overlap in scope between various non-functional requirements. For example, software performance is a broad term that includes many specific requirements like reliability and scalability.

Non-functional testing includes:

- Baseline testing
- Compatibility testing
- Compliance testing
- Documentation testing
- Endurance testing
- Load testing
- Localization testing and mangi testing
- Performance testing
- Recovery testing
- Resilience testing
- Security testing
- Scalability testing
- Stress testing
- Usability testing
- Volume testing

9.6 VALIDATION TESTING ACTIVITIES

The aim of software testing is to measure the quality of a software in terms of number of defects found in the software, the number of tests run and the system covered by the tests. These tests are carried out for both the functional and non functional attributes of the software. When bugs or defects are found with the help of testing, the bug is logged and the developer's team fixes the bug. Once the bug is fixed and testing is carried out again to ensure that the bug was indeed fixed and no new defects have been introduced in the software. With the entire cycle the quality of the software increases.

Verification and validation testing are two important tests, which are carried out on software, before it has been handed over to the customer. This makes sure, that the software testing life cycle starts early. The aim of both verification and validation is to ensure that the software product is made according to the requirements of the client and does indeed fulfill the intended purpose. So that the software product is tested thoroughly without any bias, often the job of validation testing may also be given to third party validation testing services. Therefore, validation testing is an important part of software quality assurance procedures and standards.

Software Validation Testing

While verification is a quality control process, quality assurance process carried out before the software is ready for release is known as validation testing. The validation testing goals is to validate and be confident about the software product or system, that it fulfills the requirements given by the customer. The acceptance of the software from the end customer is also a part of validation testing.

Validation testing answers the question, "Are you building the right software system". Another question, which the entire process of validation testing in software engineering answers is, "Is the deliverable fit for purpose". In other words, does the software system provide the right solution to the problem? Therefore, often the testing activities are introduced early in the software development life cycle. The two major areas, when validation testing should take place are in the early stages of software development and towards the end, when the product is ready for release. In other words, it is acceptance testing which is a part of validation testing.

Validation Testing Types

If the testers are involved in the software product right from the very beginning, then validation testing in software testing starts right after a component of the system has been developed. The different types of software validation testing are:

Component Testing

Component testing is also known as unit testing. The aim of the tests carried out in this testing type is to search for defects in the software component. At the same time, it also verifies the functioning of the different software components, like modules, objects, classes, etc., which can be tested separately.

Integration Testing

This is an important part of the software validation model, where the interaction between the different interfaces of the components is tested. Along with the interaction between the different parts of the system, the interaction of the system with the computer operating system, file system, hardware and any other software system it might interact with is also tested.

System Testing

System testing, also known as functional and system testing is carried out when the entire software system is ready. The concern of this testing is to check the behavior of the whole system as defined by the scope of the project. The main concern of system testing is to verify the system against the specified requirements. While carrying out the tester is not concerned with the internals of the system, but checks if the system behaves as per expectations.

Acceptance Testing

Here the tester especially has to literally think like the client and test the software with respect to user needs, requirements and business processes and determine, whether the software can be handed over to the client. At this stage, often a client representative is also a part of the testing team, so that the client has confidence in the system. There are different types of acceptance testing:

- Operational Acceptance Testing
- Compliance Acceptance Testing
- Alpha Testing
- Beta Testing

Often when validation testing interview questions are asked, they revolve around the different types of validation testing. The difference between verification and validation is also a common software validation testing question. Some organizations may use different terms for some of the terms given in the article above. As far as possible, I have tried to accept the alternate names as well.

9.7 BLACK BOX VS. WHITE BOX

Sr.	Black box	White box
1	Focuses on the functionality of the system	Focuses on the structure (Program) of the system
2	Techniques used are : ✓ Equivalence partitioning ✓ Boundary-value analysis ✓ Error guessing ✓ Race conditions ✓ Cause-effect graphing ✓ Syntax testing ✓ State transition testing ✓ Graph matrix	Techniques used are: ✓ Basis Path Testing ✓ Flow Graph Notation ✓ Control Structure Testing 1. Condition Testing 2. Data Flow testing ✓ Loop Testing 1. Simple Loops 2. Nested Loops 3. Concatenated Loops 4. Unstructured Loops
3	Tester can be non technical	Tester should be technical
4	Helps to identify the vagueness and contradiction in functional specifications	Helps to identify the logical and coding issues.

9.5 to 9.7 Check your Progress

Q1. Fill in the blanks

1. The concern of _____ testing is to check the behaviour of the whole system as defined by the scope of the project.
2. In _____ type of testing Tester can be non technical.

Q2. State whether True or False

1. Condition Testing is a black box testing technique.
2. In acceptance testing the tester has to literally think like the client.
3. Endurance testing is a Non functional testing technique.

9.8 SUMMARY

In functional testing functions are tested by feeding them input and examining the output and internal program structure is rarely considered. In equivalence partitioning is the process of taking all possible test cases and placing them into classes. One test value is picked from each class while testing.

Boundary value analysis' testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

A cause-effect graph is a directed graph that maps a set of causes to a set of effects. The causes may be thought of as the input to the program, and the effects may be thought of as the output. Syntax Testing uses a model of the formally-defined syntax of the inputs to a component. In structural testing an internal perspective of the system, as well as programming skills, are required and used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. Decision coverage, condition coverage and path coverage are the types of coverage testing.

Domain testing It is a field of study that defines a set of common requirements, terminology, and functionality for any software program constructed to solve a problem in that field. Non-functional testing is the testing of a software application for its non-functional requirements. While verification is a quality control process, quality assurance process carried out before the software is ready for release is known as validation testing. The validation testing goals is to validate and be confident about the software product or system, that it fulfils the requirements given by the customer. Component Testing, Integration Testing, System Testing and Acceptance Testing are the types of validation testing.

9.9 CHECK YOUR PROGRESS- ANSWERS

9.2 to 9.4

1. Boundary value analysis
2. Decision Coverage
3. Functional testing

9.5 to 9.7

Q1.

1. System
2. Black Box

Q2.

1. False
2. True
3. True

9.10 QUESTIONS FOR SELF – STUDY

1. What is functional testing?
2. Explain following functional testing techniques.

   a)Informal Reviews, b) Boundary value analysis c)Cause effect graphic, d) syntax testing
3. What is structural testing?
5. Explain coverage testing and its types.
6. What are the Non functional testing techniques?
7. Write a note on validation testing.
8. Differentiate between Black box and White box testing.

NOTES

NOTES

CHAPTER 10

Testing Specialized Systems and Applications

10.0 OBJECTIVES

Dear Friends,

After studying this chapter you will be able to -

- Describe Testing object oriented software
- Describe Testing Web based Applications
- Discuss Computer Aided Software testing tools (CAST)

10.1 INTRODUCTION

Object oriented software testing requires a different strategy from that of traditional software testing as the software is divided in classes and modules. Web application testing is also important aspect in testing as a variety of types of users are using it. CAST tools makes testing faster by providing readymade tools for the testing process.

10.2 TESTING OBJECT ORIENTED SOFTWARE

The object oriented approach to software development does not rule out the need to test the software. Object oriented software needs to be tested, though, as a result of the different architecture and unique features of the object oriented software some of the testing issues of object oriented software are different from the testing issues of the

conventional software. The unique testing issues of the object oriented software necessitate improvisation of the conventional software testing techniques and development of new testing techniques.

Class is the basic unit of testing in object oriented software. Class is a complete unit that can be isolated and tested independently. The inheritance feature of the object oriented approach allows a new class to be derived from an existing class. The inherited methods need to be retested in the derived class in context of their usage in the derived class. Dynamic binding, another powerful feature of the object oriented approach binds the relevant method at runtime. Late binding creates indefiniteness in the testing process since the method to be executed is unknown until runtime. As opposed to the waterfall model used for traditional software development, object oriented software is developed using the iterative and incremental approach. Thus, object oriented testing too becomes iterative and incremental, requiring use of regression testing techniques.

Object Oriented Software

Class is the basic building block in object oriented software. A class is a template for a set of objects that share a common data structure and common behaviour. The data structure and the behaviour of the class are represented by the data and the methods defined in the class respectively. Objects are instances of the class. An object has a state, defined behaviour and a unique identity. The state of the object is represented by the data defined in the object. Encapsulation, inheritance and polymorphism are features unique to object oriented software. Encapsulation hides the implementation details of the class. The class encapsulates the data and the internal methods.

The interaction with the class is through the set of methods that provide an interface for the class. Inheritance allows a new class (derived class) to be derived from an already existing class (base class). The derived class inherits all the features of the base class (data and methods), may override some features of the base class or define its own set of data and methods. Polymorphism allows different objects to respond to the same message in different ways. It is implemented by static or dynamic binding. Static binding occurs at compile time by the overloading of the operator or method name to provide different functionality. Dynamic binding binds the function call to the relevant method at run time. The method to be executed is unknown until runtime. Inheritance facilitates reusability of the class by extending functionality of existing classes to suit new requirements.

Class-Basic unit of testing

A unit that can be isolated and tested independently is a suitable candidate for unit testing. In conventional software, module is the basic

unit of testing. It can be isolated and tested as an independent unit. The data may be defined within the module or may be globally declared. In object oriented software, class is the basic unit of testing. The class as a standalone component is a complete unit and can be tested independently. The class encapsulates the data and has methods that operate on the data defined in the class. Since methods of the class manipulate the data defined in the class, methods cannot be isolated from the class to form a standalone unit.

Moreover, objects have state and exhibit state-dependent behaviour. The state of the object is represented by the data associated with the object. Execution of the method depends on the state of the object at the time of method invocation. Execution of the method may also alter the state of the object. Therefore, testing of the method requires observing the impact of method execution on the state of the object. The method thus cannot be isolated from the class for the purpose of testing.

Furthermore, the methods of the class are invoked via an instance of the class. The class needs to be instantiated to facilitate access to the methods of the class.

10.3 TESTING WEB BASED APPLICATIONS

Let's have first web testing checklist.

1) Functionality Testing
2) Usability testing
3) Interface testing
4) Compatibility testing
5) Performance testing
6) Security testing

1) Functionality Testing:

Test for – all the links in web pages, database connection, forms used in the web pages for submitting or getting information from user, Cookie testing.

Check all the links:

- Test the outgoing links from all the pages from specific domain under test.
- Test all internal links.
- Test links jumping on the same pages.
- Test links used to send the email to admin or other users from web pages.
- Test to check if there are any orphan pages.

- Lastly in link checking, check for broken links in all above-mentioned links.

Test forms in all pages:

Forms are the integral part of any web site. Forms are used to get information from users and to keep interaction with them. So what should be checked on these forms?

- First check all the validations on each field.
- Check for the default values of fields.
- Wrong inputs to the fields in the forms.
- Options to create forms if any, form delete, view or modify the forms.

Let's take example of the search engine project currently I am working on, in this project we have advertiser and affiliate signup steps. Each sign up step is different but dependent on other steps. So sign up flow should get executed correctly. There are different field validations like email Ids, User financial info validations. All these validations should get checked in manual or automated web testing.

Cookies testing:

Cookies are small files stored on user machine. These are basically used to maintain the session mainly login sessions. Test the application by enabling or disabling the cookies in your browser options. Test if the cookies are encrypted before writing to user machine. If you are testing the session cookies (i.e. cookies expire after the sessions ends) check for login sessions and user stats after session end. Check effect on application security by deleting the cookies. (I will soon write separate article on cookie testing)

Validate your HTML/CSS:

If you are optimizing your site for Search engines then HTML/CSS validation is very important. Mainly validate the site for HTML syntax errors. Check if site is crawlable to different search engines.

Database testing:

Data consistency is very important in web application. Check for data integrity and errors while you edit, delete, modify the forms or do any DB related functionality.

Check if all the database queries are executing correctly, data is retrieved correctly and also updated correctly. More on database testing could be load on DB, we will address this in web load or performance testing below.

2) Usability Testing:

Test for navigation:

Navigation means how the user surfs the web pages, different controls like buttons, boxes or how user using the links on the pages to surf different pages.

Usability testing includes:

Web site should be easy to use. Instructions should be provided clearly. Check if the provided instructions are correct means whether they satisfy purpose.

Main menu should be provided on each page. It should be consistent.

Content checking:

Content should be logical and easy to understand. Check for spelling errors. Use of dark colors annoys users and should not be used in site theme. You can follow some standards that are used for web page and content building. These are common accepted standards like as I mentioned above about annoying colors, fonts, frames etc. Content should be meaningful. All the anchor text links should be working properly. Images should be placed properly with proper sizes. These are some basic standards that should be followed in web development. Your task is to validate all for UI testing

Other user information for user help:

Like search option, sitemap, help files etc. Sitemap should be present with all the links in web sites with proper tree view of navigation. Check for all links on the sitemap.

"Search in the site" option will help users to find content pages they are looking for easily and quickly. These are all optional items and if present should be validated.

3) Interface Testing:

The main interfaces are:

Web server and application server interface Application server and Database server interface.

Check if all the interactions between these servers are executed properly. Errors are handled properly. If database or web server returns any error message for any query by application server then application server should catch and display these error messages appropriately to users. Check what happens if user interrupts any transaction in-between? Check what happens if connection to web server is reset in between?

4) Compatibility Testing:

Compatibility of your web site is very important testing aspect. See which compatibility test to be executed:

- Browser compatibility

- Operating system compatibility
- Mobile browsing
- Printing options

Browser compatibility:

In my web-testing career I have experienced this as most influencing part on web site testing.

Some applications are very dependent on browsers. Different browsers have different configurations and settings that your web page should be compatible with. Your web site coding should be cross browser platform compatible. If you are using java scripts or AJAX calls for UI functionality, performing security checks or validations then give more stress on browser compatibility testing of your web application. Test web application on different browsers like Internet explorer, Firefox, Netscape navigator, AOL, Safari, Opera browsers with different versions.

OS compatibility:

Some functionality in your web application is may not be compatible with all operating systems. All new technologies used in web development like graphics designs, interface calls like different API's may not be available in all Operating Systems. Test your web application on different operating systems like Windows, Unix, MAC, Linux, Solaris with different OS flavors.

Mobile browsing:

This is new technology age. So in future Mobile browsing will rock. Test your web pages on mobile browsers. Compatibility issues may be there on mobile.

Printing options:

If you are giving page-printing options then make sure fonts, page alignment, page graphics getting printed properly. Pages should be fit to paper size or as per the size mentioned in printing option.

5) Performance testing:

Web application should sustain to heavy load. Web performance testing should include:
Web Load Testing
Web Stress Testing

Test application performance on different internet connection speed. In web load testing test if many users are accessing or requesting the same page. Can system sustain in peak load times? Site should handle many simultaneous user requests, large input data from users, Simultaneous connection to DB, heavy load on specific pages etc.

Stress testing: Generally stress means stretching the system beyond its specification limits. Web stress testing is performed to break the site by giving stress and checked how system reacts to stress and how system recovers from crashes.

Stress is generally given on input fields, login and sign up areas.

In web performance testing web site functionality on different operating systems, different hardware platforms is checked for software, hardware memory leakage errors,

6) Security Testing:

Following are some test cases for web security testing:

- Test by pasting internal url directly into browser address bar without login. Internal pages should not open.
- If you are logged in using username and password and browsing internal pages then try changing url options directly. I.e. If you are checking some publisher site statistics with publisher site ID= 123. Try directly changing the url site ID parameter to different site ID which is not related to logged in user. Access should denied for this user to view others stats.
- Try some invalid inputs in input fields like login username, password, input text boxes. Check the system reaction on all invalid inputs.
- Web directories or files should not be accessible directly unless given download option.
- Test the CAPTCHA for automates scripts logins.
- Test if SSL is used for security measures. If used proper message should get displayed when user switch from non-secure http:// pages to secure https:// pages and vice versa.
- All transactions, error messages, security breach attempts should get logged in log files somewhere on web server.

Web based Applications are increasingly becoming more feature rich, important and also the most popular means for developing commercial systems. Most companies opt for developing web based software wherever possible. This helps in catering to large number of end users. The deployment of the apps (once the infrastructure is in place) is fairly easy.

The web based applications are powerful and have the ability to provide feature rich content to a wide audience spread across the globe at an economical cost.

Hence it is a daunting task to test these applications and with more and more features testing these apps is becoming even more complex.

In this article we will study the challenges faced when testing these applications

Why testing Web Applications is different?

Testing web applications is different because of many factors scenarios affecting the performance and user experience. Web applications can typically cater to a large and a diverse audience. Web Applications can also be exposed to wide range of security threats. Web applications may open up illegal points of entry to the databases and other systems holding sensitive information.

To ensure that the web application works reliably and correctly under different situations these factors need to be accounted for and tested.

Hence a lot of effort needs to put in for Test Planning and Test Design. Test Cases should be written covering the different scenarios not only of functional usage but also technical considerations such as Network speeds, Screen Resolution, etc.

For example an application may work fine on Broad Band internet users but may perform miserably for users with dial up internet connections.

Web Applications are known to give errors on slow networks, whereas they perform well on high speed connections. Web pages don't render correctly for certain situations but work okay with others.

Images may take longer to download for slower networks and the end user perception of the application may not be good.

Factors effecting Testing of Web Applications:

As mentioned above Web Applications can have a lot of variables affecting them such as:

- Numerous Application Usage (Entry – Exit) Paths are possible Due to the design and nature of the web applications it is possible that different users follow different application usage paths.

  For example in an online banking application a user may directly go to "Bill Pay" page and other users may check account balances, view previous transactions and then "Pay the Bills". Generally a large number of usage paths are possible and all are supposed to work well.

- All these Permutations and Combinations need to be tested thoroughly

- People with varying backgrounds & technical skills may use the application not all applications are self explanatory to all people.

People have varying backgrounds and may find the application hard to use. For instance a Business Intelligence application with “Drill-Down-Reports” may work out for certain users but not for others.

Although this affects the design of the applications, but these factors should be tested in usability testing of the applications

- **Intranet versus Internet based Applications**
  Intranet based applications generally cater to a controlled audience. The developers and architects can make accurate assumptions about the people accessing the apps and the hardware / software / technical specifications of the client machines.

  While it may be difficult to make similar assumptions for Internet Based Applications
  Also the intranet users can generally access the app from ‘trusted’ sources whereas for internet applications the users may need to be authenticated and the security measures may have to be much more stringent.

  Test Cases need to be designed to test the various scenarios and risks involved.

- **The end users may use different types of browsers to access the app** Typically for internet based applications users may have different Browsers when accessing the apps. This aspect also needs to be tested. If we test the app only on IE then we cannot ensure if works well on Netscape or Fire-Fox. Because these browsers may not only render pages differently but also have varying levels of support for client side scripting languages such as java-script.

- **Even on similar browsers** application may be rendered differently based on the Screen resolution/Hardware/Software Configuration

- **Network speeds:**
  Slow Network speeds may cause the various components of a Webpage to be downloaded with a time lag. This may cause errors to be thrown up.
  The testing process needs to consider this as important factor especially for Internet based Applications

- **ADA (Americans with Disabilities Act)**
  It may be required that the applications be compliant with ADA. Due certain disabilities, some of the users may have difficulty in accessing the Web Applications unless the applications are ADA compliant. The Application may need to be tested for compliance and usability.

- **Other Regulatory Compliance/Standards:**

  Depending on the nature of the application and sensitivity of the data captured the applications may have to be tested for relevant Compliance Standards. This is more crucial for Web Based Applications because of their possible exposure to a wide audience.

- **Firewalls:**
  As mentioned earlier Applications may behave differently across firewalls. Applications may have certain web services or may operate on different ports that may have been blocked. So the apps need to be tested for these aspects as well.

- **Security Aspects:**
  If the Application captures certain personal or sensitive information, it may be crucial to test the security strength of the application. Sufficient care needs to be taken that the security of the data is not compromised.

Why technology platforms affect testing?

Technology platform upon which the Web Application was built also creates different strengths and weaknesses. Different Test Automation tools & packages are available for different Technology Platforms. This can influence the Test Strategy and the way in which Test Execution is done.

Challenges in Testing Web Based Web Applications:
To ensure that sufficient Test Coverage is provided for Web Based Applications and to provide a secure, reliable application to the user the above factors need to be considered.

For internet based applications accessibility and security can be important issues.

On one hand Organizations would like to cater to their users around the world on the other hand they could end up exposing their security holes all over.

Testing could be the last chance to ensure the safety of the data and the organization.

10.4 COMPUTER AIDED SOFTWARE TESTING TOOLS (CAST)

CAST (Computer Aided Software Testing) Tools are designed to assist the testing process. They are used in various stages of SDLC.

CAST Tools support Test Management to -

1. Manage the testing process
2. Definition of what to test how to test when to test and what happened
3. Provide testing statistics / Graphs for management reporting
4. Enable impact analysis
5. Can also provide incident management
   Ex: Test Director from Mercury QA Director from Compuware

10.2 to 10.4 Check your Progress

Q1. Fill in the blanks

1. The _____ tools are designed to assist the testing process.
2. _____ is the basic unit of testing in object oriented software

Q2. State whether True or False

1. Web applications can typically cater to a large and a diverse audience.
2. In web applications numerous Application Usage (Entry – Exit) Paths are not possible.

10.5 SUMMARY

Object oriented software needs to be tested, though, as a result of the different architecture and unique features of the object oriented software some of the testing issues of object oriented software are different from the testing issues of the conventional software. . In object oriented software, class is the basic unit of testing.

Web based applications are tested for Functionality, Usability, Interface, Compatibility, Performance, Security. Testing web applications is different as Web applications can typically cater to a large and a diverse audience. Web Applications can also be exposed to wide range of security threats. Web applications may open up illegal points of entry to the databases and other systems holding sensitive information. CAST (Computer Aided Software Testing) Tools are designed to assist the testing process. They are used in various stages of SDLC.

10.6 CHECK YOUR PROGRESS- ANSWERS

10.2 to 10.4

Q1.

1. CAST (Computer Aided Software Testing)
2. Class

Q2.

1. True
2. False

10.7 QUESTIONS FOR SELF - STUDY

1. How do you test Object oriented software?
2. Explain in details Web application testing.
3. Write a note on CAST tools.

❁ ❁ ❁

NOTES

NOTES

Questions Bank

1. Explain the four categories involved in CMMI Process Area.
2. Explain PPQA Process Area in CMMI.
3. Write 5 levels of process 'maturity' that determine effectiveness in delivering quality software with CMM.
4. Write 5 stages involved in Six Sigma.
5. Explain classes for process classification.
6. Explain the need of SQA.
7. Discuss the principles of Software Quality Assurance.
8. Explain SQA tasks managed by SQA groups.
9. Write a note of software's and tools used for software quality assurance.
10. Discuss the four plans involved in Software Management.
11. Discuss the organizations involved in SQA standards.
12. Discuss the organizations involved in SQA Management standards.
13. What you understand from bathtub curve for software reliability and Hardware reliability.
14. Explain the factors of Software Failure.
15. Write a note on SRE.
16. Explain categories for software reliability matrix.
17. State the difference between software reliability prediction models and software reliability estimation models.
18. Explain the reliability measures with software modelling techniques.
19. Write a note on LOC and KLOC.
20. Discuss in detail Clean room Software Engineering.
21. Explain the terms used in software validations.
22. What is Verifications? Write the verification terms.
23. What you understand from Software Inspection'.
24. State the difference between Software verification and Software Validation.
25. List the testing objectives.
26. Explain the testing lifecycle in details.
27. Mention the phase wise activities in testing lifecycle.
28. What is a test case? Explain with an example.
29. What is the importance of test design?
30. Write a note on unit testing.
31. Define integration testing. Explain different integration strategies.
32. How system testing and acceptance testing is carried out?
33. State the Difference between i) alpha and beta testing, ii) Static & Dynamic testing, iii) Manual & Automation Testing.

34. Write a note on Tester's Workbench.
35. Discuss in detail 11 steps testing process.
36. Write a note on i) Installation testing, ii) Usability testing, iii) Regression testing.
37. Why regression testing is necessary and important?
38. Describe Performance testing.
39. Explain Load testing and stress testing in details.
40. Define security testing. Explain its measures.
41. What are reviews? Explain the benefits of it.
42. Explain the roles played in a typical formal review.
43. What a review checklist consists of?
44. Write a note on walkthroughs and inspections.
45. Discuss in details about Static analysis.
46. Describe Data flow analysis.
47. How controls flow analysis is done?
48. Write significance of cyclomatic complexity analysis.
49. Write a note on Equivalence partitioning.
50. How boundary value analysis is done.
51. Explain Cause effect graphing with the help of a diagram.
52. How syntax testing is done?
53. Explain in details about coverage testing.
54. What is domain testing?
55. Write a note on Non functional testing.
56. Explain validation testing types.
57. State Difference between Black box and White box testing.
58. Write down the testing strategy for testing object oriented software.
59. Explain the various types of tests conducted in Web testing.
60. List the factors that affect testing web applications
61. How Computer Aided Software testing tools helps management?