MAST
MARKET ALIGNED SKILLS TRAINING

AIF AMERICAN INDIA FOUNDATION

SOFTWARE TESTING TRAINING MODULE

In partnership with

Supported by:

FRANKLIN TEMPLETON INVESTMENTS

AIF AMERICAN INDIA FOUNDATION

INDIA: 1003-1005,DLF City Court, MG Road, Gurgaon 122002 Tel (91) 124 4551850 Fax (91) 124 4551888
NEW YORK: 216 E.45th Street, 7th Floor, New York, NY 10017
www.aif.org

SOFTWARE TESTING

TRAINING MODULE

About the American India Foundation

The American India Foundation is committed to catalyzing social and economic change in India, andbuilding a lasting bridge between the United States and India through high impact interventions ineducation, livelihoods, public health, and leadership development. Working closely with localcommunities, AIF partners with NGOs to develop and test innovative solutions and withgovernments to create and scale sustainable impact. Founded in 2001 at the initiative of PresidentBill Clinton following a suggestion from Indian Prime Minister Vajpayee, AIF has impacted the lives of 4.6million of India's poor.

Learn more at www.AIF.org

About the Market Aligned Skills Training (MAST) program

Market Aligned Skills Training (MAST) provides unemployed young people with a comprehensive skillstraining that equips them with the knowledge and skills needed to secure employment and succeed on thejob. MAST not only meets the growing demands of the diversifying local industries across the country, itharnesses India's youth population to become powerful engines of the economy.

AIF Team: Hanumant Rawat, Aamir Aijaz & Rowena Kay Mascarenhas

American India Foundation

10th Floor, DLF City Court, MG Road,
Near Sikanderpur Metro Station, Gurgaon 122002

216 E. 45th Street, 7th Floor New York, NY 10017
530 Lytton Avenue, Palo Alto, CA 9430

ACKNOWLEDGEMENTS

This curriculum has been developed by American India Foundation Trust in partnership with Mumbai University's Garware Institute of Career Education and Development (GICED) as part of its Market Aligned Skills Training (MAST) Program. This is supported by Franklin Templeton India as part of the program.

This curriculum is designed specifically keeping in mind the learning needs of youth having poor-to-average educational background. The objective of this curriculum is to equip AIF's MAST candidates with essentials of software testing – principles, tools, systems, applications, process of executing a software program or application with the intent of finding the software bugs.

We are grateful to Ms. Kishori Telvekar from Mumbai University's Garware Institute of Career Education and Development (GICED) for helping us in developing the content. We are also thankful to Dr. Medha Tapiawala, Director-GICED and Ms. Shilpa Borkar, Assistant Director-GICED for their continuous support and guidance in developing this curriculum.

We are also thankful to Mr Hanumant Rawat, Senior Advisor-AIF and Mr Aamir Aijaz, Program Manager-AIF to guide the content development and design team regularly.

MODULE 1

SOFTWARE

Software is a set of instructions executed by a computer which are designed to pe rform a particular task. Generally these set of instructions are termed as programs. These are of two types:

1. Application Software
2. System Software

Application Software:

These are also called end-user programs such as:

- Printout Paychecks
- Play Mortal Kombat
- Keep Track Of A Stamp Collection
- Do Your Taxes
- Generate A Fancy Newsletter
- Guide Robots
- Keep A Budget
- Draw A Flowchart
- Browse The Web
- Design A Car

The Examples of Application Software are: MS excel, word processing software like MS word, wordpad, and database software like MS access, oracle etc

System Software:

It helps computer carry out its basic task. Examples are:

- Operating Systems such as Master Control Programs, BIOS (Basic Input Output System) and Some Utilities Which are Built Into OS
- Translators such as Program Language Translators/Compliers

The Examples of Application Software are: Windows XP, Linux and Mac etc

SOFTWARE TESTING

Software testing is a process of executing a program or application with the intent of finding the software bugs. It can also be stated as the process of validating and verifying that a software program or application or product meets the business and technical requirements that guided it's design and development.

In other words, Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and to verify that the software product is fit for use.

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system is under test. Therefore:

- It is the process of executing a program with the intention of finding errors
- It can show the presence of bugs but never their absence
- A good test case is one that has a high probability of detecting an undiscovered defect, not one that shows that the program works correctly

- It is impossible to test your own program
- As the number of detected defects in a piece of software increases, the probability of the existence of more undetected defects also increases

The Need of Software Testing?

Software testing is very important because of the following reasons:

1. Software testing is required to point out the defects and errors that were made during the software development phases.
2. It's essential since it makes sure of the customer's reliability and their satisfaction in the software application.
3. It helps to ensure the quality of the product. Quality product delivered to the customers helps in gaining their confidence. (Know more about Software Quality)
4. Testing is necessary in order to provide the facilities to the customers like the delivery of high quality product or software application which requires lower maintenance cost and hence results into more accurate, consistent and reliable results.
5. Testing is required for an effective performance of software application or product.
6. It's important to ensure that the application should not result into any failures because it can be very expensive in the future or in the later stages of the development.
7. It's required to make any software business sustainable.
8. Software Testing is necessary because we all make mistakes. Some of those mistakes are unimportant, but some of them are expensive or dangerous. We need to check everything and anything we produce because things can always go wrong – humans make mistakes all the time.

Goals and Objective of Software Testing

Software testing helps in finalizing the software application or product against business and user requirements. It is very important to have good test coverage in order to test the software application completely and make it sure that it's performing well and as per the specifications. The goals and objectives of software testing are:

- Finding defects which may might have created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of quality.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the Business Requirement Specification (BRS) and System Requirement Specifications (SRS).
- To gain the confidence of the customers by providing them a quality product.

Software testing makes sure that the testing is being done properly and hence the system is ready for use. Good coverage means that the testing has been done to cover the various areas like functionality of the application, compatibility of the application with the OS, hardware and different types of browsers, performance testing to test the performance of the application and load testing to make sure that the system is reliable and should not crash or there should not be any blocking issues. It also determines that the application can be deployed easily to the machine and without any resistance. Hence the application is easy to install, learn and use

SOFTWARE BUGS

A software bug is a failure or flaw in a program that produces undesired or incorrect results. It's an error that prevents the application from functioning as it should. Most common reason for software bug is human mistakes in software design and coding.

Important Reasons for Software Bugs

- Miscommunication or no communication
- Software complexity
- Programming errors
- Changing requirements
- Time pressures
- Poorly documented code
- Software development tools
- Obsolete automation scripts
- Lack of skilled testers

1. **Miscommunication or No Communication:** Unclear requirements and misinterpretation of requirements are two major factors causing defects in software.
2. **Software Complexity:** The complexity of current software applications can be difficult to comprehend for anyone without experience in modern-day software development. Windows-type interfaces, client-server and distributed applications, data communications, enormous relational databases, and sheer size of applications have all contributed to the exponential growth in software/system complexity.

3. **Programming Errors:** Inexperienced programmers or programmers without proper domain knowledge can introduce simple mistakes while coding.
4. **Changing Requirements:** The customer may not understand the effects of changes, or may understand and request them anyway – redesign, rescheduling of engineers, effects on other projects, work already completed that may have to be redone or thrown out, hardware requirements that may be affected, etc.
5. **Time Pressures:** When deadlines loom and the crunch comes, mistakes will be made.
6. **Poorly Documented Code:** It's tough to maintain and modify code that is badly written or poorly documented; the result is *software bugs.*
7. **Software Development Tools :** Visual tools, class libraries, compilers, scripting tools, etc. often introduce their own bugs or are poorly documented, resulting in added bugs.
8. **Obsolete Automation Scripts :** Writing automation scripts takes lot of time especially for complex scenarios.
9. **Lack of Skilled Testers:** Having skilled testers with domain knowledge is extremely important for success of any project. But appointing all experienced testers is not possible for all companies. Domain knowledge and the tester's ability to find defects can produce high quality software. Compromise on any of this can result in buggy software.

Software Testing Process

- Planning and Control
- Analysis and Design
- Implementation and Execution
- Evaluating exit criteria and Reporting
- Test Closure activities

1. **Planning and Control**

Test planning has following major tasks:

- To determine the scope and risks and identify the objectives of testing
- To determine the test approach.
- To implement the test policy and/or the test strategy.
- To the required test resources like people, test environments, PCs, etc.
- To schedule test analysis and design tasks, test implementation, execution and evaluation.
- To determine the Exit criteria we need to set criteria such as Coverage criteria.

Test control has the following major tasks:

- To measure and analyze the results of reviews and testing
- To monitor and document progress, test coverage and exit criteria
- To provide information on testing
- To initiate corrective actions
- To make decisions

2. **Analysis and Design:**

Test analysis and Test Design has the following major tasks:

- To review the test basis
- To identify test conditions
- To design the tests
- To evaluate testability of the requirements and system
- To design the test environment set-up and identify and required infrastructure and tools

3. **Implementation and Execution:**

During test implementation and execution, we take the test conditions into test cases and procedures and other testware such as scripts for automation, the test environment and any other test infrastructure

4. **Test implementation**

Test Implementation has the following major task:

- To develop and prioritize our test cases by using techniques and create test data for those tests.
- To create test suites from the test cases for efficient test execution.
- To implement and verify the environment.

Test execution has the following major task:

- To execute test suites and individual test cases following the test procedures
- To re-execute the tests that previously failed in order to confirm a fix. This is known as confirmation testing or re-testing
- To log the outcome of the test execution and record the identities and versions of the software under tests. The test log is used for the audit trial.
- To compare actual results with expected results.
- To report discrepancies as Incidents where there are differences between actual and expected results

5. **Evaluating Exit criteria and Reporting**

Exit criteria comes into picture, when

- Maximum test cases are executed with certain pass percentage
- Bug rate falls below certain level
- When achieved the deadlines

Evaluating exit criteria has the following major tasks

- To check the test logs against the exit criteria specified in test planning
- To assess if more test are needed or if the exit criteria specified should be changed
- To write a test summary report for stakeholders.

6. **Test Closure activities**

Test closure activities are done when software is delivered. The testing can be closed for the other reasons also like:

- When all the information has been gathered which are needed for the testing
- When a project is cancelled.
- When some target is achieved.
- When a maintenance release or update is done.

Test closure activities have the following tasks:

- To check which planned deliverables are actually delivered and to ensure that all incident reports have been resolved
- To finalize and archive testware such as scripts, test environments, etc. for later reuse
- To handover the testware to the maintenance organization. They will give support to the software
- To evaluate how the testing went and learn lessons for future releases and projects

Test Initiation Criteria

Timing: As soon as we have a software requirement

Objective: To trap requirements-related defects as early as they can be identified

Approach: Test requirements

We prevent incorrect requirements from being incorporated in the design and implementation where they will be more difficult and expensive to find and correct. To pass through the quality gateway and be included in the requirmetns specification, a requirement must pass a number of tests. These tests are concerned with ensuring that the requirements are accurate, and do not causes problems by being unsuitable for design and implementation stages later in the project.

Test completion Criteria

Timing: As soon as we have a software requirements

Objective: To trap requirements-related defects as early as they can be identified

Approach: Test requirements

Participants in Testing

Timing: As soon as we have a software requirements

Objective: To trap requirements-related defects as early as they can be identified

Approach: Test requirements

Best practices of Testing

- Learn to analyze your test results thoroughly
- Learn to maximize the test coverage
- To ensure maximum test coverage break your application under test (AUT) into smaller functional modules
- Write your test cases in requirement analysis and design phase itself
- Make your test cases available to developers prior to coding
- If possible identify and group your test cases for regression testing
- Programmers should not test their own code
- Go beyond requirement testing

Do not ignore the test result. Testers will be respected if they not only log the bugs but also provide solutions. Though 100 percent test coverage might not be possible still you can always try to reach near it. To ensure maximum test coverage break your application under test (AUT) into smaller functional modules. Write test cases on such individual unit modules. Also if possible break these modules into smaller parts. While writing test cases, write test cases for intended functionality first i.e. for valid conditions according to requirements.

Then write test cases for invalid conditions. This will cover expected as well unexpected behaviour of application under test. Write your test cases in requirement analysis and design phase itself. This way you can ensure all the requirements are testable. Make your test cases available to developers prior to coding. Don't keep your test cases with you waiting to get final application release for testing, thinking that you can log more bugs. Let developers analyze your test cases thoroughly to develop quality application. This will also save the re-work time. If possible identify and group your test cases for regression testing. This will ensure quick and effective manual regression testing. Programmers should not test their own code. As discussed in our previous post, basic unit testing of developed application should be enough for developers to release the application for testers. But you (testers) should not force developers to release the product for testing. Let them take their own time. Everyone from lead to manger know when the module/update is released for testing and they can estimate the testing time accordingly. This is a typical situation in agile project environment.

Challenges in Testing

- Complete testing is impossible
- Testers misallocate resources because they fall for the company's process myths
- Test groups operate under multiple missions, often conflicting, rarely articulated
- Test groups often lack skilled programmers, and a vision of appropriate projects that would keep programming testers challenged

Complete testing is impossible. There is no simple answer for this. Therefore testers live and breathe tradeoffs. Testers misallocate resources because they fall for the company's process myths. Testers have to rely on their wits, not on someone else's compliance with an (alleged but unrealistic) process. Test groups operate under multiple missions, often conflicting, rarely articulated. We pick our tests to conform to our testing mission. Test groups often lack skilled programmers, and a vision of appropriate projects that would keep programming testers challenged

Test Methods

Static vs. Dynamic Testing

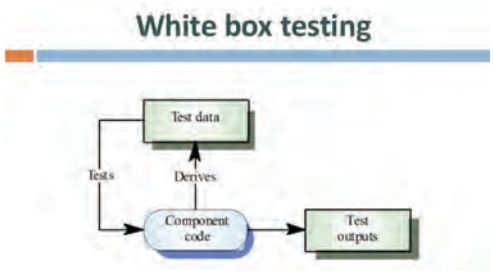
Static vs. dynamic testing: Static testing is often implicit, as proofreading, plus when programming tools/text editors check source code structure or compilers (pre-compilers) check syntax and data flow as static program analysis. Dynamic testing takes place when the program itself is run.

The Box Approach

- White-box testing
- Black-box testing
- Visual testing
- Grey-box testing

White-Box Testing:

White-box testing (also known as clear box testing, glass box testing, transparent box testing and structural testing, by seeing the source code) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases.



Black-Box Testing:

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it.

Visual Testing:

The aim of visual testing is to provide developers with the ability to examine what was happening at the point of software failure by presenting the data in such a way that the developer can easily find the information she or he requires, and the information is expressed clearly

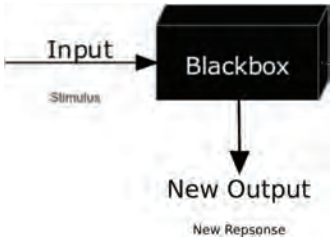
Grey-Box Testing:

Grey-box testing (American spelling: gray-box testing) involves having knowledge of internal data structures and algorithms for purposes of designing tests, while executing those tests at the user, or black-box level. The tester is not required to have full access to the software's source code. Manipulating input data and formatting output do not qualify as grey-box, because the input and output are clearly outside of the "black box" that we are calling the system under test.

BLACK BOX TEST DESIGN TECHNIQUES

- Decision table testing
- All-pairs testing
- Equivalence partitioning
- Boundary value analysis
- Cause–effect graph
- Error guessing

- State transition testing
- Use case testing
- User story testing
- Domain analysis
- Combining technique



Decision Table Based Testing:

This approach is the most rigorous one and is ideally implemented when the number of combinations of actions is taken under varying conditions.

Equivalence Class Partitioning:

This technique is used to reduce the number of possible inputs to small yet effective inputs. Used to test an application exhaustively and avoid redundancy of inputs, it is done by dividing inputs into classes and getting value from each class.

Boundary Value Analysis:

The most commonly used black box testing technique, Boundary Value Analysis or BVA is used to find the error in the boundaries of input values rather than the center.

Cause-Effect Graphing Technique:

This technique considers a system's desired external behavior only. It helps in selecting test cases which relate Causes to Effects to create test cases. In the aforementioned statement, Cause implies a distinct input condition which results in internal change in a system while Effect implies an output condition brought by a combination of causes.

Error Guessing:

The success of this technique is solely dependent on the experience of the tester. There are no tools and techniques as such, but one can write test cases either while reading the document or while encountering an undocumented error during the testing.

Black Box Test Advantages

- Efficient when used on large systems
- Testing is balanced and unprejudiced, Since the tester and developer are independent of each other
- Tester can be non-technical
- Detailed functional knowledge of system for the tester is not necessary
- Tests are done from an end user's point of view, because the end user should accept the system. (This testing technique is sometimes also called Acceptance testing)
- Testing helps to identify vagueness and contradictions in functional specifications
- Test cases can be designed as soon as the functional specifications are complete

Black Box Test Disadvantages

- Tests can be redundant if already run by the software designer
- Test cases are extremely difficult to be designed without clear and concise specifications
- Testing of every possible input stream is not possible because it is time-consuming and this would eventually leave many program paths untested
- Results might be overestimated at times
- Cannot be used for testing complex segments of code

WHITE BOX TEST DESIGN TECHNIQUES

White box testing, also known as structural testing or code-based testing, is a methodology which ensures and validates a software application's mechanisms, internal framework, and objects and components.

This method of testing not only verifies a code as per the design specifications, but also uncovers an application's vulnerabilities.

It is also known as transparent box, glass box, and clear box testing as it clearly visualizes the software's internal mechanisms for a software engineering team.

During the white box testing phase, a code is run with preselected input values to validate the preselected output values. If a mismatch is found, it implies that the software application is marred by a bug. This process also involves writing software code stubs and drivers

The most important part in the white box testing method is the **code coverage analysis** which empowers a software engineering team to find the area in a code which is unexecuted by a given set of test case thereby, helping in improving a software application's quality. There are different techniques which can be used to perform the code coverage analysis. Some of these are:

Statement Coverage: This technique is used to test every possible statement at least once. Cantata++ is the preferred tool when using this technique.

Decision Coverage: This includes testing every possible decision condition and other conditional loops at least once. TCAT-PATH, supporting C, C++, and Java applications, is the go-to tool when this technique is followed.

Condition Coverage: This makes one time code execution mandatory when all the conditions are tested.

Decision/Condition Coverage: This is a mixed technique which is implemented to test all the Decision / Condition coverage at least once while the code is executed.

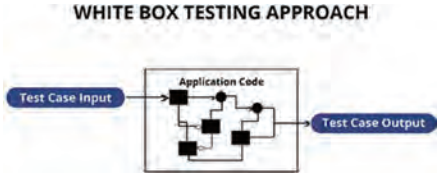
Multiple Condition Coverage: In this type of white box testing technique, each entry point of a system has to be executed at least once.

White Box Test Advantages

- Code optimization by revealing hidden errors
- Transparency of the internal coding structure which is helpful in deriving the type of input data needed to test an application effectively
- Covers all possible paths of a code thereby, empowering a software engineering team to conduct thorough application testing
- Enables programmer to introspect because developers can carefully describe any new implementation
- Test cases can be easily automated
- Gives engineering-based rules to stop testing an application

White Box Test Disadvantages

- A complex and expensive procedure which requires the adroitness of a seasoned professional, expertise in programming and understanding of internal structure of a code
- Updated test script required when the implementation is changing too often
- Exhaustive testing becomes even more complex using the white box testing method if the application is of large size
- Some conditions might be untested as it is not realistic to test every single one
- Necessity to create full range of inputs to test each path and condition make the white box testing method time-consuming
- Defects in the code may not be detected or may be introduced considering the ground rule of analyzing each line by line or path by path



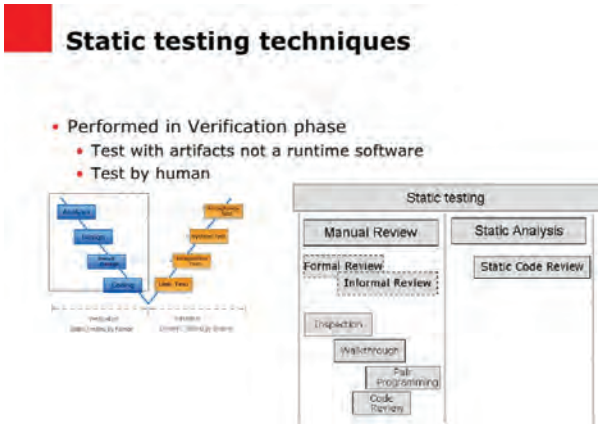
DIFFERENCES BETWEEN WHITE BOX TESTING AND BLACK BOX TESTING

Basis of Differentiation	White Box Testing	Black Box Testing
Performed at Levels	White Box Testing is applicable to lower levels like Unit and Integration levels	This method of testing is mainly applicable to higher levels of testing like Acceptance and System
Performed by	Software Developers	Independent Software Testers
Programming Knowledge	Professionals performing this method to test the functionality of an application should be knowledgeable in multiple programming languages	No such criteria is necessary when it comes to black box testing
Basis of Test Case Design	Detailed design documents are essential to design test cases when testing an application using the white box testing method	Requirement specifications documents are enough to design test cases for black box testing
Focus of Testing	White box testing focuses on how the inputs are carried out and the code of a system	This type of testing focuses on what is being carried out and the functionality of a system
Level of Complexity	Extreme	Moderate

THE STATIC TEST TECHNIQUES

Static Testing is type of testing in which the code is not executed. It can be done manually or by a set of tools. This type of testing checks the code, requirement documents and design documents and puts review comments on the work document. When the software is non –operational and inactive, we perform security testing to analyse the software in non-runtime environment. With static testing, we try to find out the errors, code flaws and potentially malicious code in the software application. It starts earlier in development life cycle and hence it is also called verification testing. Static testing can be done on work documents like requirement specifications, design documents, source code, test plans, test scripts and test cases, web page content. It includes:

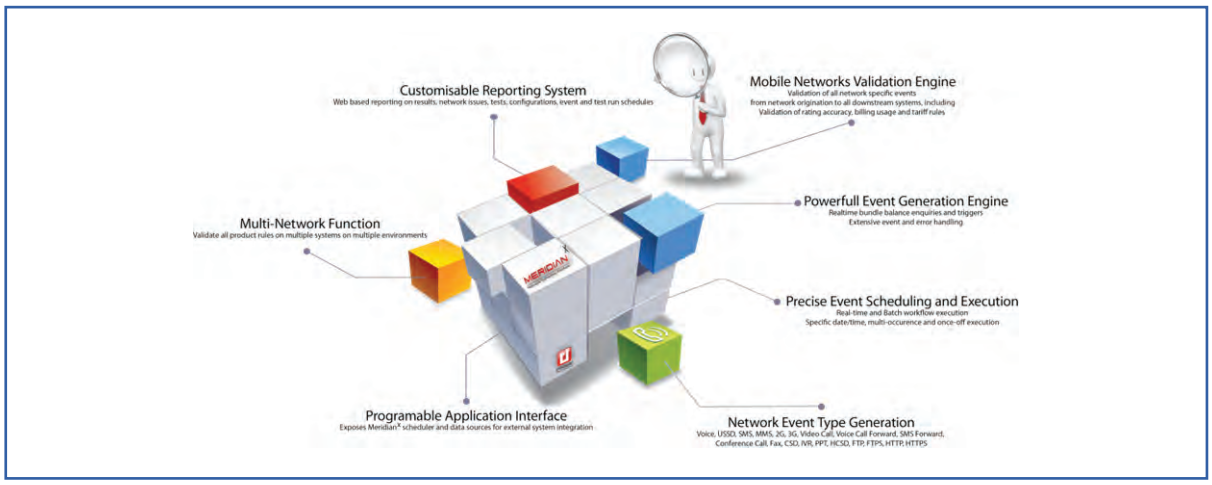
- **Inspection:** Here the main purpose is to find defects. Code walkthroughs are conducted by moderator. It is a formal type of review where a checklist is prepared to review the work documents.
- **Walkthrough:** In this type of technique a meeting is lead by author to explain the product. Participants can ask questions and a scribe is assigned to make notes.
- **Technical reviews:** In this type of static testing a technical round of review is conducted to check if the code is made according to technical specifications and standards. Generally the test plans, test strategy and test scripts are reviewed here.
- **Informal reviews:** Static testing technique in which the document is reviewed informally and informal comments are provided.



THE DYNAMIC TEST TECHNIQUES

Dynamic testing is done when the code is in operation mode. Dynamic testing is performed in runtime environment. When the code being executed is input with a value, the result or the output of the code is checked and compared with the expected output. With this we can observe the functional behaviour of the software, monitor the system memory, CPU response time, performance of the system. Dynamic testing is also known as validation testing , evaluating the finished product. Dynamic testing is of two types: Functional Testing and Non functional testing.

- **Unit Testing:** Testing of individual modules by developers. The source code is tested in it
- **Integration Testing:** Testing the interface between different modules then they are joined
- **System Testing:** Testing performed on the system as a whole
- **Acceptance Testing:** Testing done from user point of view at user's end



THE DIFFERENCE BETWEEN STATIC & DYNAMIC TEST TECHNIQUES

Static Testing	Dynamic Testing
1. Static Testing is white box testing which is done at early stage if development life cycle. It is more cost effective than dynamic testing	1. Dynamic Testing on the other hand is done at the later stage of development lifecycle.
2. Static testing has more statement coverage than dynamic testing in shorter time	2. Dynamic Testing has less statement stage because it is covers limited area of code
3. It is done before code deployment	3. It is done after code deployment
4. It is performed in Verification Stage	4. It is done in Validation Stage
5. This type of testing is done without the execution of code.	5. This type of execution is done with the execution of code.
6. Static testing gives assessment of code as well as documentation.	6. Dynamic Testing gives bottlenecks of the software system.
7. In Static Testing techniques a checklist is prepared for testing process	7. In Dynamic Testing technique the test cases are executed.
8. Static Testing Methods include Walkthroughs, code review.	8. Dynamic testing involves functional and nonfunctional testing

Fundamentals of Software Quality

- Quality
- Quality Views
- Quality and Productivity
- Employee involvement in Quality

Quality:

- High levels of user satisfaction and low level of defects often associated with low complexity: Tom McCabe
- Quality is another name of consistently meeting customers requirements, cost delivery schedule and services offered

- Quality must be measurable and should predictable
- Quality must be well defined and should be Quantified
- Quality objectives should be achievable and traceable

Quality Views:

A. Customer's View

- Delivering the right product
- Satisfying customers needs
- Meeting the customers' expectations
- Treating every customer with integrity, courtesy, respect

Since Quality is perceived value by the customers, their view reflects the following factors

1. Delivering the right product

   The product, which is delivered, should be the one addressing customer requirements. The product should contain all the features mentioned in requirement specifications

2. Satisfy customer need

   The product may be satisfying all requirements, but may still fall short of his needs. The financial planning system may have all the features for creating a financial plan and perform analysis, but if the analysis reports do not provide the customer any learning, then it may not be satisfying his needs

3. Meeting customer expectations

   Customer expectation may fall into two categories Viz. Expressed expectation documented in requirement specifications and implied expectations which may not form a part of requirement specifications

B. Suppliers' View

- Doing the right thing
- Doing it the right way
- Doing it right the first time
- Doing it in time

Quality Means:

Consistently meeting customer needs in terms of

- Requirements
- Cost
- Delivery schedule
- Service

It is the process of meeting the customer requirements first time and every time, which is fit for use to perform its intended functions, with reasonable cost and within time

Software Quality:

Simple Tips:

- Aim at customer delight
- Have measurable objectives
- Understand requirements accurately; have a through traceability of ALL requirements
- Implement the plan-do-check-act cycle in each phase
- Detect and remove defects as early as possible: prevention is better than cure
- Systematic change control and version control must exist
- Follow easy to use standards/conventions for naming, commenting, coding and documentation
- Start with compiling and analyzing simple metrics

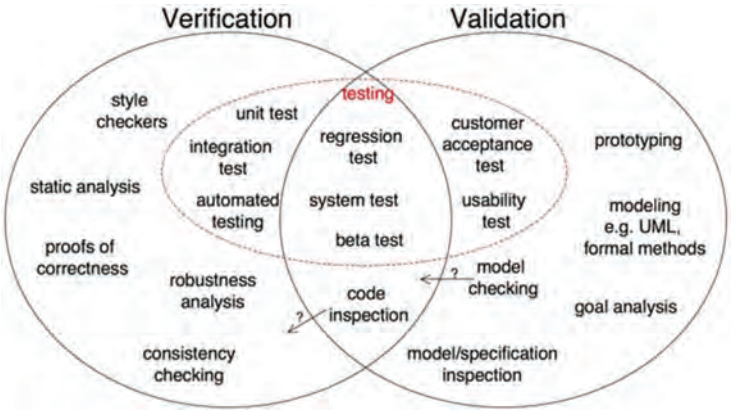
MODULE 2

SOFTWARE VERIFICATION AND VALIDATION

- Verification
- Validation
- V Model

Objectives:

- To Understand the significance of verification and validation
- To Clearly illustrate techniques of verification and validation
- To Explain the guidelines associated with various techniques
- To Explain the V model



Verification

- It is a disciplined approach to evaluate whether a software product fulfils the requirements or conditions imposed on them
- It is also called static testing
- It is done by systematically reading the contents of a software product with the intention of detecting defects
- It helps in identifying not only presence of defects but also their location
- A filter is applied at various points during the SDLC to purify the product as it progresses through various phases
- It makes sure that the product is designed to deliver all functionality to the customer.
- It is done at the starting of the development process. It includes reviews and meetings, walkthroughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications.

Suppose you are building a table. Here the verification is about checking all the parts of the table, whether all the four legs are of correct size or not. If one leg of table is not of the right size it will imbalance the end product. Similar behavior is also noticed in case of the software product or application. If any feature of software product or application is not up to the mark or if any defect is found then it will result into the failure of the end product. Hence, verification is very important. It takes place at the starting of the development process.

Advantages of Software Verification

- Verification helps in lowering down the count of the defect in the later stages of development.
- Verifying the product at the starting phase of the development will help in understanding the product in a better way.
- It reduces the chances of failures in the software application or product

Walkthrough

- An informal process, initiated by the author of a software product to a colleague for assistance in locating defects and for suggesting improvements
- Normally not planned
- Author explains the product observations
- Colleague comes out with observations
- Author provides clarification if required
- Author notes down relevant points and takes corrective actions

During the walkthrough meeting the presenter 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

For example, the programmer who has written the code will formally present the code fuction by function or line by line. The reviewers then question the programmer for any doubts/queries

Inspection

A thorough , word-by-word checking of a software product (or part of product) with the intention of:

- Locating defects
- Confirming traceability of relevant requirements
- Checking for conformance to relevant standards and conventions

A typical inspection session takes around 90-120 minutes and typically they detect 30-70% of logic design and coding errors. The objective of the inspection is to find errors in the program, thus improving the quality of the work

Pre-conditions for inspection

- A precise specification must be available
- Team members must be familiar with the organisation standards
- Syntactically correct code must be available
- An error checklist should be prepared

The process of inspection is:

- System overview presented to inspection team
- Code and associated documents are distributed to inspection team in advance
- Inspection takes place and discovered errors are noted
- Modification are made to repaired discovered errors
- Re-inspection may or may not be required

Involves 5 Major Roles:

- Author
- Moderator
- Reader
- Recorder
- Inspector

Author

An author is the person who originally constructed the work product. The author will address specific questions that arise concerning the content of the work product. The author will be ultimately responsible for updating the work product after the inspection e.g a programmer

Moderator

The moderator is responsible for ensuring that the discussions proceed on the productive lines and that the participants focus their attention on funding errors.

Reader

The reader is responsible for documenting all defects that arise from the inspection meeting. This documentation will include where the defect was found

Inspector

All of the inspection Team individuals are also considered to play the inspector role, independent of other roles assigned. The inspector role is responsible for analyzing and detecting defects within the work product

Example –Inspection checklist

Control Flow

- Will each loop terminate
- Do/END statements match
- Will program terminate

Data Declaration

- Are all variable declared
- Are arrays and strings initialized properly

Calculation

- Division by Zero
- Non arithmetic variables

Input/Output

- Files opened before use and closed after use
- Are I/O Errors handled

Case Study Inspection:

Requirement

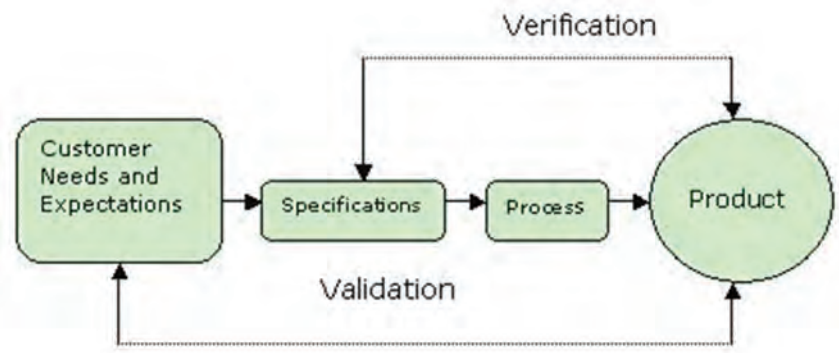
- The program shall convert centigrade value to Fahrenheit and vice versa starting from 0 to 300 with interval of 20
- The program shall print all centigrade and Fahrenheit values

The above slide listed the requirement for developing a program to be developed in C language. This requirement will be the input for developing to generate appropriate code

Review

- Follow-up: a subsequent examination of a product for the purpose of monitoring earlier changes
- Formal or official examination
- A process in which one or more persons checks changed documents or data to determine if the changes are correct

- An analysis undertaken at a fixed point in time to determine the degree to which stated objectives have been reached. This is generally used as a basis for decision making, including updating plans



Validation

- It is a disciplined approach to evaluate whether the final, as built software product fulfils its specific intended use
- It's also called as dynamic testing
- It's done systematically by the presence of defects, not their location
- It is necessary to demonstrate not just that the software is doing what it is supposed to do , but also is not doing what it is not supposed to do
- Method: Testing each software product at each phase of life cycle using test plan, test cases
- Validation is nothing but a process of finding out if the product being built is right?
- The software product should functionally do what it is supposed to, it should satisfy all the functional requirements set by the customer/user
- Validation is done during or at the end of the development process in order to determine
- Whether the product satisfies specified requirements. Validation is usually performed by dynamic testing, testing with execution on a computer
- Validation and verification processes go hand in hand, but visibly validation process starts after verification process ends
- 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/Code testing, integration validation/ integration testing, functional validation/functional testing and system /user acceptance testing/validation

Validation – Levels of Testing

- Unit testing
- Integration testing
- System testing
- User acceptance testing

Unit is the smallest pice of software that can be tested in isolation. Testing of such units is called unit testing

The testing conducted to check whether modules work properly when integrated is integration testing. Generally will be functional testing

System testing is testing done in order check whether the system works properly as whole in a simulated test environment

Acceptance testing is a testing conducted by customer in order to verify that system works properly in his environment and his needs are satisfied

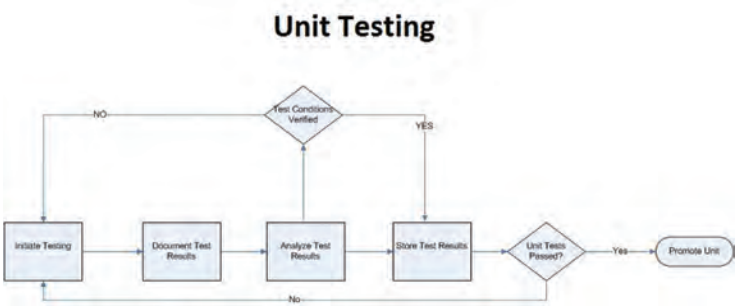
Unit Testing

- A unit is the smallest testable part of an application like functions, classes, procedures, interfaces. Unit testing is a method by which individual units of source code are tested to determine if they are fit for use.
- Unit tests are basically written and executed by software developers to make sure that code meets its design and requirements and behaves as expected.
- The goal of unit testing is to segregate each part of the program and test that the individual parts are working correctly.
- This means that for any function or procedure when a set of inputs are given then it should return the proper values. It should handle the failures gracefully during the course of execution when any invalid input is given.
- A unit test provides a written contract that the piece of code must assure. Hence it has several benefits.
- Unit testing is basically done before integration

Method Used for unit testing: White Box Testing method is used for executing the unit test.

When Unit testing should be done? : Unit testing should be done before Integration testing.

By whom unit testing should be done? : Unit testing should be done by the developers.



Advantages of Unit Testing

Issues are found at early stage. Since unit testing are carried out by developers where they test their individual code before the integration. Hence the issues can be found very early and can be resolved then and there without impacting the other piece of codes.

Unit testing helps in maintaining and changing the code. This is possible by making the codes less interdependent so that unit testing can be executed. Hence chances of impact of changes to any other code gets reduced.

it also helps in reducing the cost of bug fixes, since the bugs are found early in unit testing hence. Just imagine the cost of bug found during the later stages of development like during system testing or during acceptance testing.

- Unit testing helps in simplifying the debugging process. If suppose a test fails then only latest changes made in code needs to be debugged.

Integration Testing

- Integraption testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.
- Also after integrating two different components together we do the integration testing. As displayed in the image below when two different modules ‘Module A’ and ‘Module B’ are integrated then the integration testing is done

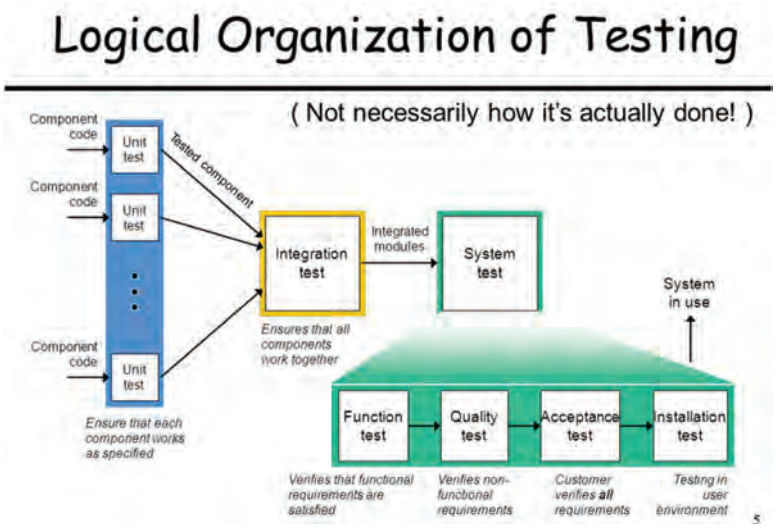
- Integration testing is done by a specific integration tester or test team.
- Integration testing follows four approach known as

  Top Down' approach

  Bottom Up' approach

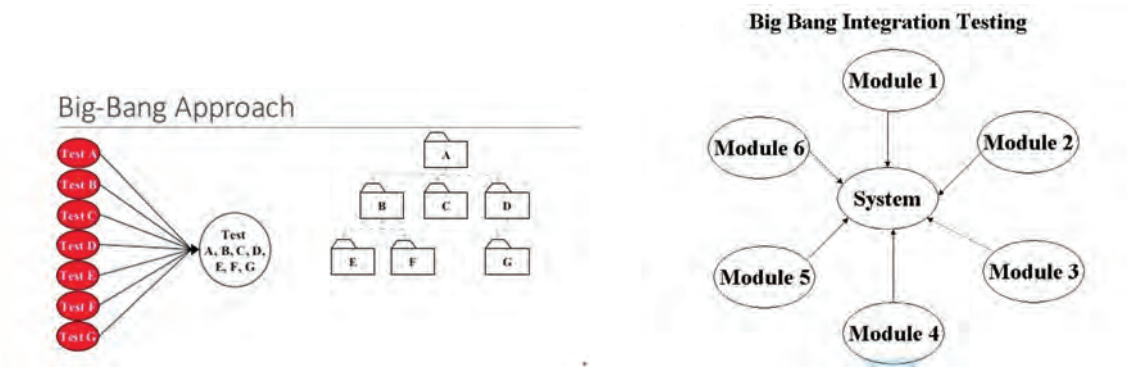
  Big bang approach

  Critical Part first



Big Bang Integration Testing

In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole. As per the below image all the modules from 'Module 1' to 'Module 6' are integrated simultaneously then the testing is carried out.



- In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole.
- In this approach individual modules are not integrated until and unless all the modules are ready.
- In Big Bang integration testing all the modules are integrated without performing any integration testing and then it's executed to know whether all the integrated modules are working fine or not.
- This approach is generally executed by those developers who follows the 'Run it and see' approach.
- Because of integrating everything at one time if any failures occurs then it become very difficult for the programmers to know the root cause of that failure.

- In case any bug arises then the developers has to detach the integrated modules in order to find the actual cause of the bug.

Advantage of Big Bang Integration:

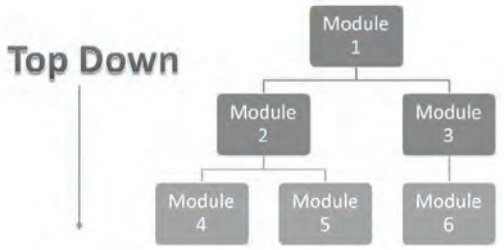
- Big Bang testing has the advantage that everything is finished before integration testing starts.

Disadvantages of Big Bang Integration:

- The major disadvantage is that in general it is very time consuming
- It is very difficult to trace the cause of failures because of this late integration.
- The chances of having critical failures are more because of integrating all the components together at same time.
- If any bug is found then it is very difficult to detach all the modules in order to find out the root cause of it.
- There is high probability of occurrence of the critical bugs in the production environment

Top-down Integration Testing

- Testing takes place from top to bottom, following the control flow or architectural structure (e.g. starting from the GUI or main menu). Components or systems are substituted by stubs. Below is the diagram of 'Top down Approach':



Advantages of Top-Down approach:

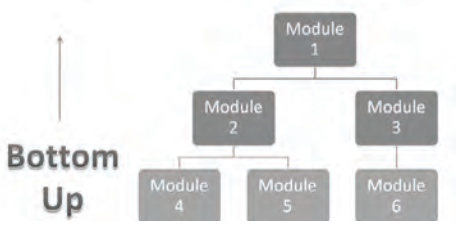
- The tested product is very consistent because the integration testing is basically performed in an environment that almost similar to that of reality
- Stubs can be written with lesser time because when compared to the drivers then Stubs are simpler to author.

Disadvantages of Top-Down approach:

- Basic functionality is tested at the end of cycle

Bottom-up Integration Testing

Testing takes place from the bottom of the control flow upwards. Components or systems are substituted by drivers. Below is the image of 'Bottom up approach':

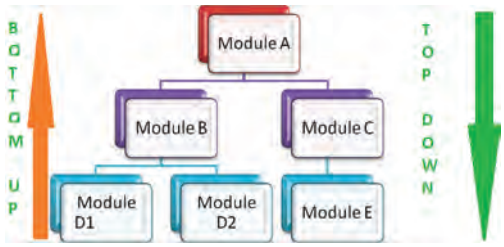


Advantage of Bottom-Up approach:

- In this approach development and testing can be done together so that the product or application will be efficient and as per the customer specifications.

Disadvantages of Bottom-Up approach:

- We can catch the Key interface defects at the end of cycle
- It is required to create the test drivers for modules at all levels except the top control



System Integration Testing

- It tests the interactions between different systems and may be done after system testing
- It verifies the proper execution of software components and proper interfacing between components within the solution.
- The objective of SIT Testing is to validate that all software module dependencies are functionally correct and that data integrity is maintained between separate modules for the entire solution.
- As testing for dependencies between different components is a primary function of SIT Testing, this area is often most subject to Regression Testing.

System Testing

- In system testing the behavior of whole system/product is tested as defined by the scope of the development project or product.
- It may include tests based on risks and/or requirement specifications, business process, use cases, or other high level descriptions of system behavior, interactions with the operating systems, and system resources.
- System testing is most often the final test to verify that the system to be delivered meets the specification and its purpose.
- System testing is carried out by specialists testers or independent testers.
- System testing should investigate both functional and non-functional requirements of the testing.

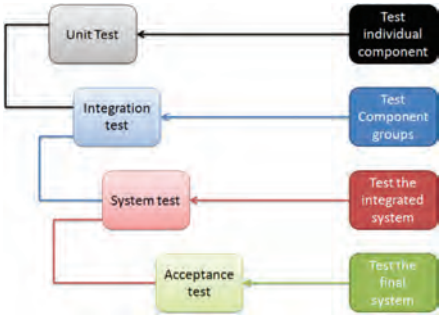
Acceptance Testing

- After the system test has corrected all or most defects, the system will be delivered to the user or customer for acceptance testing.
- Acceptance testing is basically done by the user or customer although other stakeholders may be involved as well.
- The goal of acceptance testing is to establish confidence in the system.
- Acceptance testing is most often focused on a validation type testing.
- Acceptance testing may occur at more than just a single level, for example:
  - A **Commercial Off the shelf (COTS)** software product may be acceptance tested when it is installed or integrated.
  - **Acceptance testing of the usability of the component** may be done during component testing.

Acceptance testing of a new functional enhancement may come before system testing.

The types of acceptance testing are:

- **Acceptance test:** focuses mainly on the functionality thereby validating the fitness-for-use of the system by the business user. The user acceptance test is performed by the users and application managers.
- **Operational Acceptance test:** also known as Production acceptance test validates whether the system meets the requirements for operation. In most of the organization the operational acceptance test is performed by the system administration before the system is released. The operational acceptance test may include testing of backup/restore, disaster recovery, maintenance tasks and periodic check of security vulnerabilities.
- **Contract Acceptance testing:** It is performed against the contract's acceptance criteria for producing custom developed software. Acceptance should be formally defined when the contract is agreed.
- **Compliance acceptance testing:** It is also known as regulation acceptance testing is performed against the regulations which must be adhered to, such as governmental, legal or safety regulations.

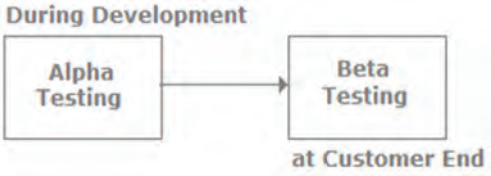


Alpha Testing

Alpha testing is one of the most common software testing strategies used in software development. Its specially used by product development organizations.

- This **test takes place at the developer's site.** Developers observe the users and note problems.
- Alpha testing is testing of an application when development is about to complete. Minor design changes can still be made as a result of alpha testing.
- Alpha testing is typically performed by a group that is independent of the design team, but still within the company, e.g. in-house software test engineers, or software QA engineers.
- Alpha testing is final testing before the software is released to the general public. It has two phases:
  - In the **first phase** of alpha testing, the software is tested by in-house developers. They use either debugger software, or hardware-assisted debuggers. The goal is to catch bugs quickly.
  - In the **second phase** of alpha testing, the software is handed over to the software QA staff, for additional testing in an environment that is similar to the intended use.

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing



Beta Testing

- It is also known as field testing. It takes place at **customer's site**. It sends the system to users who install it and use it under real-world working conditions.
- A beta test is the second phase of software testing in which a sampling of the intended audience tries the product out. (Beta is the second letter of the Greek alphabet.) Originally, the term *alpha test* meant the first phase of testing in a software development process. The first phase includes unit testing, component testing, and system testing. Beta testing can be considered "pre-release testing.
- The goal of beta testing is to place your application in the hands of real users outside of your own engineering team to discover any flaws or issues from the user's perspective that you would not want to have in your final, released version of the application.

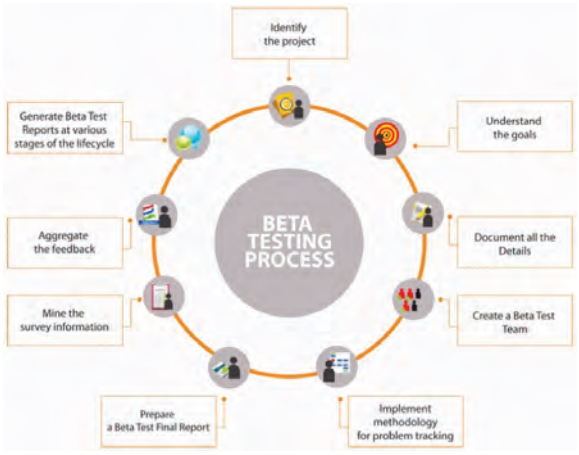
Open and closed Beta:

Developers release either a **closed beta** or an **open beta;**

- Closed beta versions are released to a select group of individuals for a user test and are invitation only, while
- Open betas are from a larger group to the general public and anyone interested. The testers report any bugs that they find, and sometimes suggest additional features they think should be available in the final version.

Advantages of Beta Testing:

- You have the opportunity to get your application into the hands of users prior to releasing it to the general public.
- Users can install, test your application, and send feedback to you during this beta testing period.
- Your beta testers can discover issues with your application that you may have not noticed, such as confusing application flow, and even crashes.
- Using the feedback you get from these users, you can fix problems before it is released to the general public.
- The more issues you fix that solve real user problems, the higher the quality of your application when you release it to the general public.
- Having a higher-quality application when you release to the general public will increase customer satisfaction.
- These users, who are early adopters of your application, will generate excitement about your application.



V Testing concept

- Life cycle testing: Continuous testing throughout the SDLC
- Goes hand-in-hand with formalized system development process
- Need to plan the testing activities parallel with the SDLC phase

Software testing has now become an integral part of the software development process. The formal V model is a software development and testing model which helps identify need of planning and design of testing in the early stage of development process. One arms of V model is our conventional waterfall model whereas another arm of V shows corresponding testing methodology required at each phase of development process.

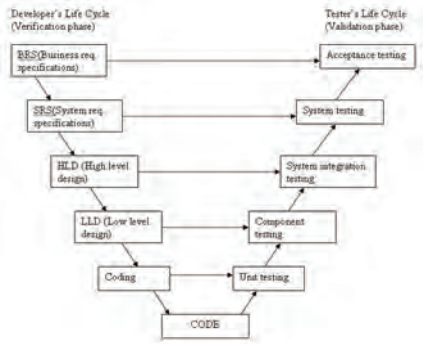
The major benefit of having both SDLC along with STLC is that, testing and development both get equal importance in the process and forces management for their commitment, attention and planning of required resources equally for both.

Another important advantage is that, output from the development phase can be tested or reviewed by testing team to validate the testability

The last but not least, early planning of testing process significantly reduces non-conformities which otherwise not detected at their respective phases.

V Model

V- model means Verification and Validation model. Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development in **V-model.**



Requirements like BRS and SRS begin the life cycle model just like the waterfall model. But, in this model before development is started, a **system test** plan is created. The test plan focuses on meeting the functionality specified in the requirements gathering.

The high-level design (HLD) phase focuses on system architecture and design. It provide overview of solution, platform, system, product and service/process. An **integration test** plan is created in this phase as well in order to test the pieces of the software systems ability to work together.

The low-level design (LLD) phase is where the actual software components are designed. It defines the actual logic for each and every component of the system. Class diagram with all the methods and relation between classes comes under LLD. Component tests are created in this phase as well.

The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Coding: This is at the bottom of the V-Shape model. Module design is converted into code by developers.

Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

When to use the V-model:

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

MODULE 3

SPECIAL TESTS

- Requirements testing
- Usability testing
- Configuration testing
- Compatibility testing
- Installation testing
- Localization & Internationalization

Other special tests

- Regression testing
- Re-testing
- Smoke testing
- Sanity testing

Specialized systems

- Client server systems
- Web based systems
- Desktop systems

Requirement Testing

Requirements-based testing is a testing approach in which test cases, conditions and data are derived from requirements. It includes functional tests and also non-functional attributes such as performance, reliability or usability. In this testing, the process starts from requirement phase and continues until operations and maintenance phase.

The objectives of requirement testing are:

- To implement the user requirement
- To maintain the correctness
- To ensure the processing complies with organisation's policies and procedures

The methods used to meet the objectives are:

- Creation of a matrix to determine whether all requirements are fulfilled
- Use of checklist to verify whether system meets organizational policies and governmental regulations

The usage of software testing is:

- To ensure that system performs correctly
- To ensure that correctness can be sustained for a considerable period of time
- To ensure that the system can be tested for correctness through all phases of SDLC but in case of reliability the programs should be in place to make system operational

Installation Testing

Installation 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.

It determines whether:

- The Installation procedure is documented
- The personnel are trained in installation process
- The methodology for migration from old system to new system is documented

Installation tests will check the installation and configuration procedure as well as any missing dependences. The ideal installation might simply appear to be run a setup program, the generation of that setup program itself and its efficacy in a variety of machine and operating system environments can require extensive testing before it can be used with confidence. In distributed systems, particularly where software is to be released into an already live target environment (such as an operational web site) installation (or deployment as it is sometimes called) can involve database schema changes as well as the installation of new software. A factor that can increase the organizational requirements of such an exercise is need to synchronize the data in the test deployment environment with that in the live environment with minimum disruption to live operation

Usability Testing

Usability testing, a non-functional testing technique that is a measure of how easily the system can be used by end users. It is difficult to evaluate and measure but can be evaluated based on the level of Skill required to learn/use the software. It is performed to test the ease of using an application

It determines:

- How simple it is to understand application usage
- How easy it is to execute an application process
- Direct observation of people using the system
- Usability surveys
- Beta tests

In designing for usability, following are the important questions to be asked:

- How easy it is for a user who has never seen the product before to carry out basic tasks?
- How easy it will be for the user who has used the product before to remember how to carry out the same tasks?
- How effective and efficient will it be for user who has used the product before to quikly carry out frequent used tasks?
- How good is the user experience in using the product?

It checks for human factor problems like:

- Are outputs meaningful?
- Are error diagnostics straightforward?
- Does GUI have conformity of Syntax, convetions, format, style abbreviations?
- Is it easy to use?
- Is there an exit option in all choices?

It should not:

- Annoy intended user in function or speed
- Take control from the user without indicating when it will be returned

It should:

- Provide on-lone help or user manual
- Be consistent in its function and overall design

User Interface Testing

It is perform to check how user-friendly the application is. Therefore the interface testing is the process of testing a product's graphical user interface to ensure it meets its specifications. This is normally done through the use of a variety of test cases.

It determines whether:

- Appropriate input help is displayed on screen
- Correct messages are displayed when an error is encountered
- Columns have meaningful names
- Navigation within the application is easy

It should be performed without the assistance of the system personnel. Effective UI design is one that provides the highest usability to the users.

The following design elements are important in this testing process

1. The tester should be aware of existing color schemes and standards
2. The tester should be aware of object sizing such as height and width of various screen items. Example – standard for scrollbars is that they are required to be eleven pixels wide. If one application has a nine pixel wide scrollbar when the standard is set to eleven, then it is an inconsistency and it should be logged
3. Item and object title spacing are other important standards that should not be ignored. If an item title is not correctly placed, then another application inconsistency is the result
4. The tester should also be familiar with title naming conventions. For example
5. If all description field have the naming convention of descar and the name in the application is description then this situation qualifies as bug
6. Last, but not least on the GUI design standard hierarchy of importance are font sizes and types. If the client prefers all of their all of their titles to be upper case Times New Roman forn at a ten point pitch and the application has lower case titles of the same font, then a discrepancy has arisen

Performance Testing

Performance testing is the process of determining the speed or effectiveness of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions.

Load

Load determines response times of critical business processes and transaction to determine whether they are within documented expectations

Stress

Stress determines the load under which a system fails, and how it fails

Configuration Testing

Configuration testing is the method of testing an application with multiple combinations of software and hardware to find out the optimal configurations that the system can work without any flaws or bugs. It determines the effect of adding or modifying the resources such as:

- Memory
- Disk drives
- CPU
- Network card

The purpose of configuration testing is to test for compatibility issues, determine minimal and optimal configuration of hardware and software, and determine the effect of adding or modifying resources such as: memory, disk drives and CPU

Configuration testing is easy with most automated testing tools because scripts are reusable and can be easily run on different platforms. Testing different system configurations or different systems by running the same workload and comparing response time

Compatibility Testing

Compatibility testing is a type of software testing used to ensure compatibility of the system/application/website built with various other objects such as other web browsers, hardware platforms, users (in case if it's very specific type of requirement, such as a user who speaks and can read only a particular language), operating systems etc. This type of testing helps find out how well a system performs in a particular environment that includes hardware, network, operating system and other software etc.

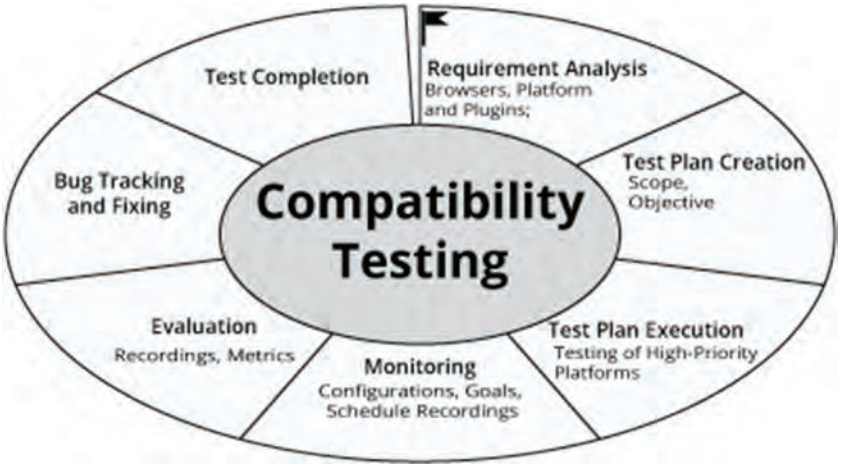
It is similar to multi-platform testing. It is performed to ensure that application functions properly on multiple system configurations, more significantly where any browser can be used to access the application.

Usage

Compatibility testing determines if an application under supported configurations performs as expected with various combinations of hardware and software flavours and releases

Example:

Configuration system would validate that a web application supports a browser (of specific configuration), compatibility testing would determine which vendors under the same configuration are compatible with the web application



Localization and Internationalization

It is a type of non-functional testing.

Internationalization is a process of designing a software application so that it can be adapted to various languages and regions without any changes.

Whereas Localization is a process of adapting internationalized software for a specific region or language by adding local specific components and translating text

Localization:

The aspect of development and testing relating to the translation of the software and its presentation to the end user. This includes translating the program, choosing appropriate icons and graphics, and other cultural considerations. It also may include translating the programs help files and the documentation

Internationalization

The aspect of development and testing related to handling foreign text and data within a program. This includes sorting, importing text and data, correct handling of currency and date and time formats, string parsing, upper

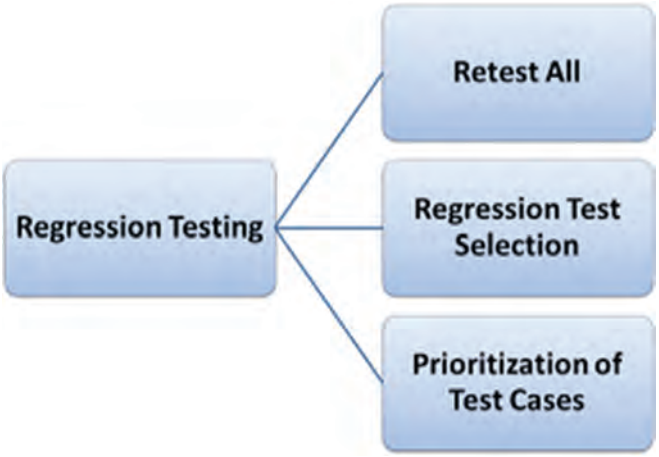
and lower case handling, and so forth. It also includes the task of separating strings from the source code, and making sure that the foreign language strings have enough space in your user interface to be displayed correctly

Regression Testing

Regression testing is a type of software testingwhich verifies that software which was previously developed and tested still performs the same way after it was changed or interfaced with other software. Changes may include software enhancements, patches, configuration changes, etc.

When any modification or changes are done to the application or even when any small change is done to the code then it can bring unexpected issues. Along with the new changes it becomes very important to test whether the existing functionality is intact or not. This can be achieved by doing the regression testing.

- The purpose of the regression testing is to find the bugs which may get introduced accidentally because of the new changes or modification.
- During confirmation testing the defect got fixed and that part of the application started working as intended. But there might be a possibility that the fix may have introduced or uncovered a different defect elsewhere in the software. The way to detect these 'unexpected side-effects' of fixes is to do regression testing.
- This also ensures that the bugs found earlier are NOT creatable.
- Usually the regression testing is done by automation tools because in order to fix the defect the same test is carried out again and again and it will be very tedious and time consuming to do it manually.
- During regression testing the test cases are prioritized depending upon the changes done to the feature or module in the application. The feature or module where the changes or modification is done that entire feature is taken into priority for testing.
- This testing becomes very important when there are continuous modifications or enhancements done in the application or product. These changes or enhancements should NOT introduce new issues in the existing tested code.
- This helps in maintaining the quality of the product along with the new changes in the application.



Types of Regression Testing:

There are four types of regression testing techniques. They are as follows:

- **Corrective Regression Testing:** Corrective regression testing can be used when there is no change in the specifications and test cases can be reused.
- **Progressive Regression Testing:** Progressive regression testing is used when the modifications are done in the specifications and new test cases are designed.

- **Retest-All Strategy:** The retest-all strategy is very tedious and time consuming because here we reuse all test which results in the execution of unnecessary test cases. When any small modification or change is done to the application then this strategy is not useful.
- **Selective Strategy:** In selective strategy we use a subset of the existing test cases to cut down the retesting effort and cost. If any changes are done to the program entities, e.g. functions, variables etc., then a test unit must be rerun. Here the difficult part is to find out the dependencies between a test case and the program entities it covers.

Advantages of Regression testing:

- It helps us to make sure that any changes like bug fixes or any enhancements to the module or application have not impacted the existing tested code.
- It ensures that the bugs found earlier are NOT creatable.
- Regression testing can be done by using the automation tools
- It helps in improving the quality of the product.

Disadvantages of Regression testing:

- If regression testing is done without using automated tools then it can be very tedious and time consuming because here we execute the same set of test cases again and again.
- Regression test is required even when a very small change is done in the code because this small modification can bring unexpected issues in the existing functionality.

Smoke Testing

Smoke testing is a type of software testing which ensures that the major functionalities of the application are working fine. This testing is also known as 'Build Verification testing'. It is a non-exhaustive testing with very limited test cases to ensure that the important features are working fine and we are good to proceed with the detailed testing.

The term **'smoke' testing** is originated from the **hardware testing**, where a device when first switched on is tested for the smoke or fire from its components. This ensures that the hardware's basic components are working fine and no serious failures are found.

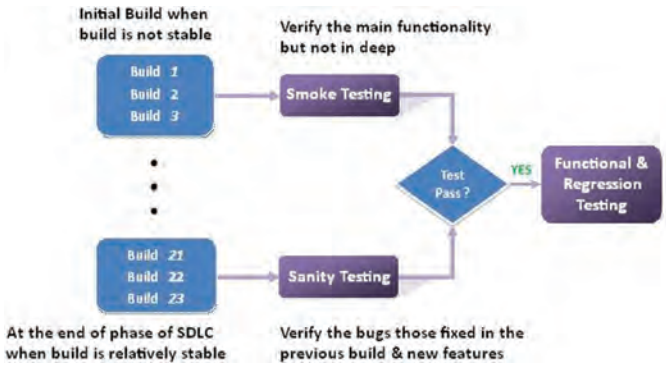
Similarly, when we do smoke testing of an application then this means that we are trying to ensure that there should NOT be any major failures before giving the build for exhaustive testing.

Advantages of Smoke testing

- It helps in finding the bugs in the early stage of testing.
- It helps in finding the issues that got introduced by the integration of components.
- It helps in verifying the issues fixed in the previous build are NOT impacting the major functionalities of the application.
- Very limited number of test cases is required to do the smoke testing.
- Smoke testing can be carried out in small time.

Disadvantages of Smoke testing

- Smoke testing does not cover the detailed testing.
- It's a non-exhaustive testing with small number of test cases because of which we not are able to find the other critical issues.



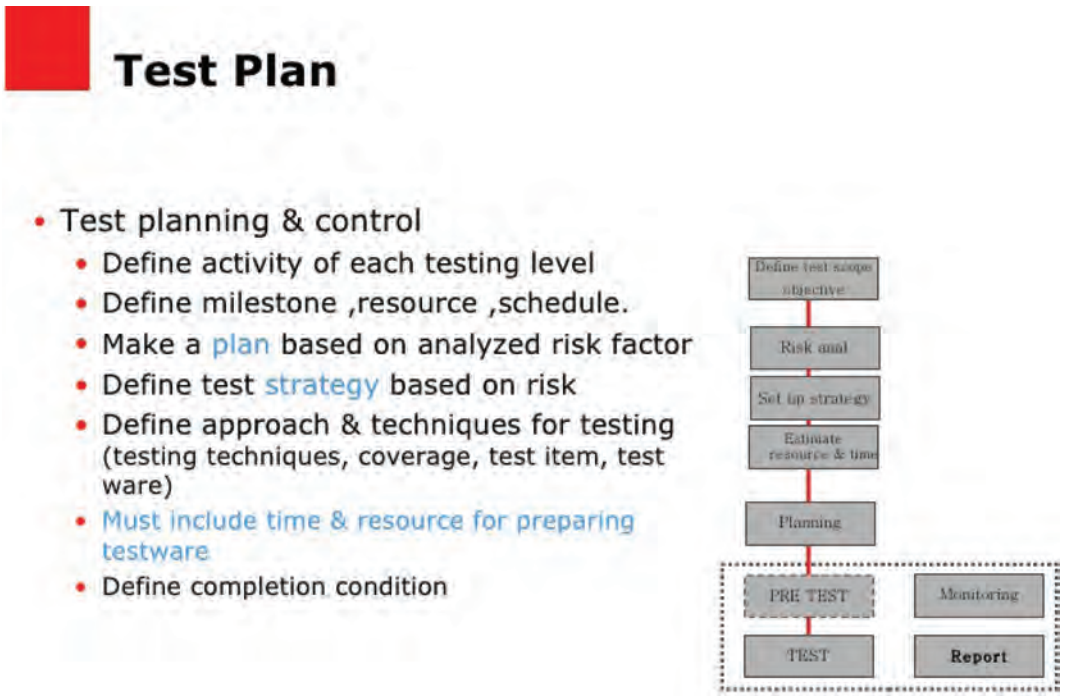
MODULE 4

TESTING PLANNING

A Software Test Plan is a document describing the testing scope and activities. It is the basis for formally testing any software/product in a project. It discusses about objectives, scope, different criteria, approach, individual roles and responsibilities, schedules, risks and their mitigation, approvals, item pass/fall criteria et.al

Why to plan anything?

- You plan those things that you know are more complicated than what you can do
- You plan things for which order and completeness are important
- You Plan things when you know that current industry standards are tested and released software averages more than ten significant bugs per 1000 lines of code.
- You plan to think ahead and planning your testing is one way to cut that down



Test Plan

It is a contract between the testers and the project team/users describing the role of project team/ users describing the role of testing in the project. This stage is where all of the work involved in planning and set-up pays off. It describes the way in which we will show our customers that the software works correctly. Also it explains who does the testing, why the tests are performed, how the tests are conducted and when the tests are scheduled

A test plan states what the items to be tested are, at what level they will be tested, what sequence they are to be tested in, how the test strategy will be applied to the testing of each item describes the test environment

A test plan should ideally be organisation-wide, being applicable to all of the organisation software development. The objective of each test plan is to provide a plan for verification, by testing the software, if the software produced fulfils the functional or design statements of the appropriate software specification. In the case of acceptance testing and system testing, this generally means the functional specification

The test plan answers such questions as

- What is being tested?
- What are pass/fail criteria?
- When will each test occur?

- What hardware and software environment is required?
- What features must be tested?
- What are the responsibilities of individuals and organizations involved in the project?

Test Plan Template

- Test plan identifier
- Introduction
- Test scope
- Test objectives
- Assumptions
- Risk Analysis
- Strategy
- Features to be tested/not to be tested
- Roles and responsibilities
- Test schedule and resources
- Test environment
- Communication approach
- Test tools

Test Plan Identifier

It is an unique identifier for the test plan with version for which we are preparing the plan

Introduction

"Overview of System X"

"Purpose of this document"

This document intends to serve as the draft test approach for the business systems development project. Preparation for this test consists of three major stages

1. The test approach sets the scope of system testing the overall strategy to be adopted the activities to be completed the general resources required and the method and processes to be used to test the release. It also details the activities dependencies and effort required to conduct the system test
2. Test planning details the activities dependencies and effort required to conduct the system test
3. Test conditions/cases documents the tests to be applied, the data to be processed the automated testing coverage and the expected results

"Formal Reviewing"

There will be several formal review points before and during system test. This is a vital element in achieving a quality product

Test Scope

Answers two important questions _____

- What will be covered in the test
- What will not be converted in the test

Includes-

- Functional or structural requirements
- System interfaces
- Application documentation

Test Objectives

- Can be simply called as a Testing Goal
- Guides development of test cases, procedures and test data
- Tester and project manager can gauge testing progress
- Enhance communication both within and outside of the project team by helping to define the scope of the testing effort

Example-Scope

The collegiate sports paging system will be unit tested and system tested. Unit tests will address functional quality, while system testing will address issues of scalability and performance

The interaction of the subsystems will be tested as follows:

Content management to paging

Content management to reporting

The following systems interfaces will be tested:

Collegiate sports paging system to existing WebNewsOnLine Web Server

Collegiate sports paging system to paging gateways

The most critical testing will be that of load and performance testing. This will be addressed as follows—

We will create a test scenario that will generate increasing numbers of pages up to 200,000

We will also create a test scenario that has new content arriving at the system at the rate of one item every 20 seconds. Lastly, we will simulate increasing concurrent subscriber loads up to 200,000. It is a statement of what the tester is expected to accomplish or validate during a specific testing activity. Test objective includes high level description of expected results in measurable terms and is prioritized

Assumption

Document test prerequisites, which if not met could have negative impact on the test

Example:

- Skill level of resources
- Test budget
- State of application at start of testing
- Tools available

List out realistic assumptions which are used as basis for test planning

Risk analysis

Document test risk and their possible impact on the test effort

Example:

- New technology
- New test automation tool
- Sequence and increments of code delivery
- Availability of application test resources

Identify what software is to be tested and what the critical areas are, such as delivery of a third party product

- New version of interfacing software
- Ability to use and understand a new package/tool etc
- Extremely complex functions
- Modifications to components with a past history of failure
- Poorly documented modules or change requests

There are some inherent software risks such as complexity; these need to be identified:

- Safety
- Multiple interfaces
- Impacts on client
- Government regulations and rules

Strategy

The strategy should identify major test activities and the methodology employed to execute the test activities

- Overall approach of testing
  - Black/White Box
- Types/levels of tests
  - Unit Integration
  - Security/Stress
- Major activities
  - Test Environment
  - Test Data

Features to be tested/not to be tested:

Features to be tested:

This is listing of what is to be tested from the USERS viewpoint of what the system does. This is not a technical description of the software, but a USERS view of the functions

Features not to be tested:

This is a listing of what is NOT to be tested from both the users' viewpoint of what the system does and a configuration management/version control view

Roles & Responsibilities

It defines who is responsible for each stage of testing and generates a responsibility matrix.

Role: ***Test designer***

Responsibilities:

- To generate test plan
- To generate test model
- To evaluate effectiveness of test efforts

Role: ***Tester***

Responsibilities:

- To execute tests
- To log results
- To document change requests

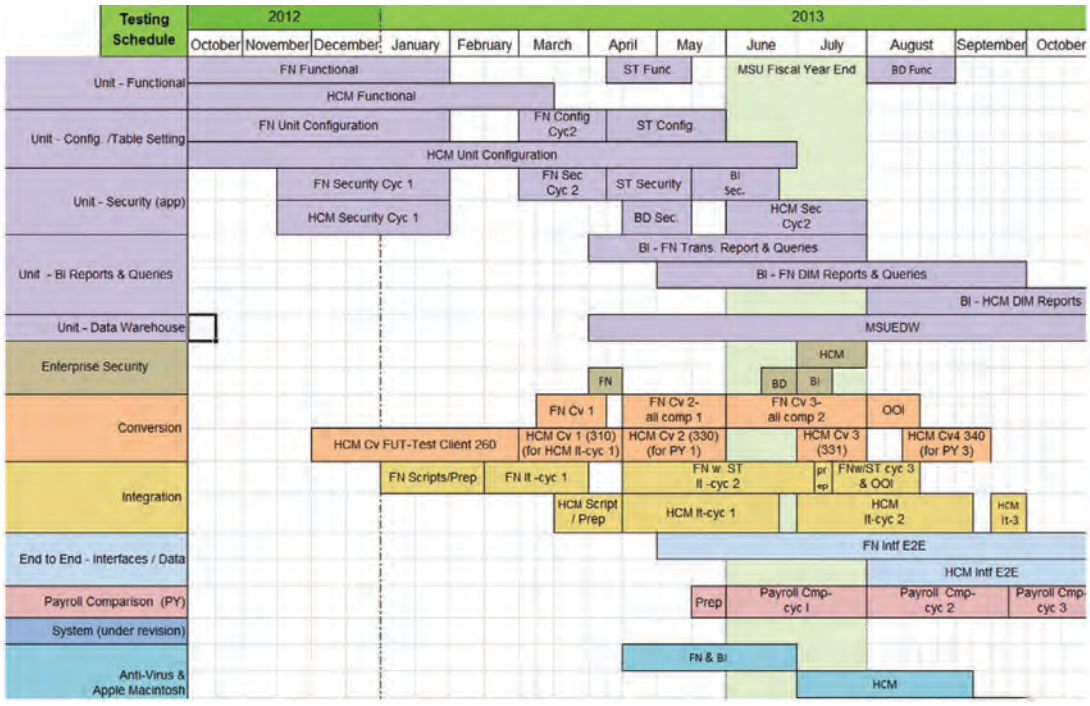
Test Schedule and Resources

Test Schedule:

A test schedule includes the testing steps or tasks, the target start and end dates, and responsibilities. It should also describe how the test will be reviewed, tracked, and approved. It also includes:

- Major test activities
- Sequence of tests
- Dependence on other project activities
- Initial estimates for each activity

The Planned Resources include People, Tools and Facilities.



Suspension/Resumption Criteria

Suspension criteria specify the criteria to be used to suspend all or a portion of the testing activities while resumption criteria specify when testing can resume after it has been suspended.

- Unavailability of external dependent systems during execution.
- When a defect is introduced that cannot allow any further testing.
- Critical path deadline is missed so that the client will not accept delivery even if all testing is completed.
- A specific holiday shuts down both development and testing

System Integration Testing in the Integration environment may be resumed under the following circumstances:

- When the external dependent systems become available again
- When a fix is successfully implemented and the testing team is notified to continue testing
- The contract is renegotiated with the client to extend delivery
- The holiday period ends

Suspension criteria assumes that testing cannot go forward and that going backward is also not possible. A failed build would not suffice as you could generally continue to use the previous build. Most major or critical defects would also not constituted suspension criteria as other areas of the system could continue to be tested.

Test Environment:

For test environment, key area to set up includes:

- System and applications
- Test data
- Database server
- Front end running environment
- Client operating system
- Browser
- Hardware includes Server Operating system
- Network
- Documentation like reference documents/configuration guides/installation guides/ user manuals

A testing environment is a setup of software and hardware on which the test team will conduct the testing

Few challenges while setting up the test environment are:

- Remote environment
- Combined usage between teams
- Elaborate setup time
- Ineffective planning for resource usage for integration
- Complex test configuration

Communication Approach:

- Formal and informal meetings
- Working sessions
- Communication tools to be used such as electronic bulletin boards, mails, intranet sites etc.
- Escalation procedures

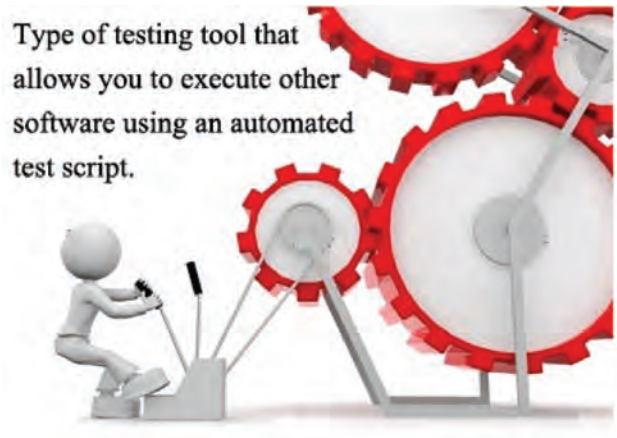
- Miscellaneous items such as project contact lists, meeting audiences

Test Tools

Tools are those which are needed to support the testing process. Tools are used for:

- Test management
- Configuration management
- Test script development
- Automated test tools
- Stress/load testing tools
- Defect tracking tools

Any tool that will be needed to support the testing process should be included here. The information outlined here cannot usually all be completed at once, but is captured in greater levels of detail as the project progresses through the life cycle.



Importance of Test Plan

A test plan is a detailed document that outlines the test strategy, testing objectives, resources (manpower, software, and hardware) required for testing, test schedule, test estimation and test deliverables. It serves as a blueprint to conduct software testing activities as a defined process which is minutely monitored and controlled by the test manager. Making Test Plan has multiple benefits:

- Test Plan helps us determine the effort needed to validate the quality of the application under test
- It helps people outside the test team such as developers, business managers, customers understand the details of testing.
- Test Plan guides our thinking. It is like a rule book, which needs to be followed.
- Important aspects like test estimation, test scope, test strategy are documented in Test Plan, so it can be reviewed by Management Team and re-used for other projects.

How to write a test plan:

You already know that making a Test Plan is the most important task of Test Management Process. Follow the seven steps below to create a test plan

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

Guidelines to develop a test plan:

- Start early
- Keep the test plan flexible
- Frequently review the test plan
- Keep the test plan concise and readable
- Calculate the planning effort
- Spend the time to do a complete test plan

MODULE 5

TEST DESIGN

A test design is the act of creating and writing test suites for testing software. Test analysis and identifying test conditions gives us a generic idea for testing which covers quite a large range of possibilities. But when we come to make a test case we need to be very specific. In fact now we need the exact and detailed specific input. But just having some values to input to the system is not a test, if you don't know what the system is supposed to do with the inputs, you will not be able to tell that whether your test has passed or failed.

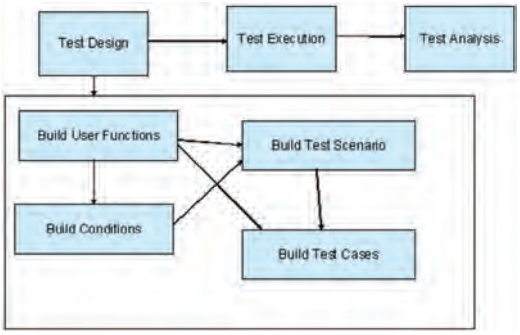
More than the act of testing, the act of designing tests is one of the best bug presenters known. The thinking that must be done to create a useful test can discover and eliminate bugs before they are coded-indeed, test-design thinking can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest

Objectives:

- To understand the significance of test design
- To know the details of test case writing
- To write good test cases
- To identify the common test case mistakes
- To explore the contents of the test case
- To develop testing techniques like equivalence class, boundary value analysis etc.
- To Manage test data

Importance of Test Design:

- Foundation to design and develop test script
- Greater confidence in quality
- Completeness of test
- Estimation of test effort



Test Design Essentials:

- Test cases cover all features
- There is a balance between normal, abnormal, boundary and environment test cases
- There is a balance between black box and white box testing
- There is a balance between functional tests and non functional tests
- Finally, documented test cases

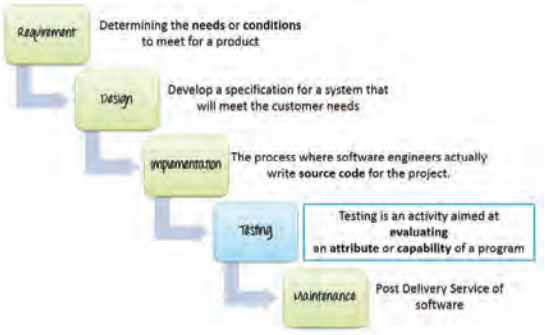
Effective test design is a key to success for many organizations and for the projects. Desiging test cases gives you a chance to analyzed the specification from different angles

- You can repeat the same test cases
- Someone can execute the test cases for you
- Validate the quality of test cases
- Estimate the quality of target software early on

What is a good test case?

1. It should be accurate and do what it is intended to test.
2. No unnecessary steps should be included in it.
3. It should be reusable.
4. It should be traceable to requirements.
5. It should be compliant to regulations.
6. It should be independent i.e. you should be able to execute it in any order without any dependency on other test cases.
7. It should be simple and clear, any tester should be able to understand it by reading once.

Now keeping in mind these characteristics you can write good and effective test cases



TIPS for writing good test cases

Tests only one thing

Always make sure that your test case tests only one thing, if you try to test multiple conditions in one test case it becomes very difficult to track results and errors.

Organize your test cases consistently

You can organize your test cases in many ways however you should always follow the same pattern to organize you test cases.

Write independent test cases

Your test cases should not have dependency on other test cases, i.e you should be able to execute your test case individually with having dependency on other test cases.

Write small test cases

Always mention purpose of each test case clearly in test case.

Following things should be kept in mind while writing the good test cases:

- Testability-Easy to test
- Use active case, do this, do that
- System displays this, does that
- Simple conversational language
- Exact, consistent names of fields, not generic
- Dont explain windows basics
- Order of cases follows business scenarios

How to write good test cases

Improving testability of test cases

- The definition of testability is easy to test –accurately.
- It takes to execute the test and whether the tester has to get clarification during the testing process
- Accurately means that if the tester follows the directions, the result of pass or fail will be correct

Improving testability with language

- Test steps should be written in active case. Tell the tester what to do.

Example

Do this. Do that

Navigate to the shopping cart page

Compare the items in the cart with the screen capture

Click on <OK>

It should always be clear whether the tester is doing something or the system is doing it.

If tester reads, “The button is pressed” does that mean he or she should press the button, or does it mean to verify that the system displays it as already presses? One of the fastest ways to confuse a tester is to mix up actions and results. In figure 1, actions always go on the left side, results on the right side. What the tester does is always an action. What the system displaysor does is always a result.

Step	Action	Expected Result
1	Enter new name and address. Press <OK>	Displays screen 008 new name details
2	Fill all blank with natural data. Make screen grab	Displays screen 005 maintenance. Press <OK>
3	Click on <inquiry> button	Displays screen 009 inquiry details
4	Enter name from screen grab. Press <OK>	Displays screen 010 record detail
5	Compare record detail with screen grab	All details match exactly

An excellent test case satisfies the following criteria

- Reasonable probability of catching an error
- Exercises an area of interest
- Does not do unnecessary things
- Neither too simple nor too complex
- Not redundant with other tests
- Makes failures obvious
- Allows isolation and identification of errors

Test cases as development assets have a life beyond testing. They represent a complete picture of how the software works written in plain English. Even if the focus is destructive they must also prove that all business scenarios work as required. Often the cases are written for testers who are the business users so they use real world language and terms. A set of use cases has tremendous value to others who are working to learn or sell the software

- Business users
- Technical writers
- Help desk technicians
- Trainers
- Sales and marketing staff
- Web administrators