EDISON Software Development Centre
+7 (499) 500 14 94
http://www.edsd.com
market@edsd.com

Quality Assurance

1. TESTING BASICS

1.1. What is Quality Assurance?

The other technical terminology for Quality Assurance is Verification and Validation. To put in simple words, Verification speaks on "Are we building the right system?" and Validation speaks on "Are we building the system right?". QA is an umbrella activity that is applied throughout the development process. But Quality as the American Heritage Dictionary defines is, "A Characteristic or attribute of something..."

1.2. What is Testing by the way?

In general, testing is finding out how well something works. In terms of human beings, testing tells what level of knowledge or skill has been acquired. In computer hardware and software development, testing is used at key checkpoints in the overall process to determine whether objectives are being met. For example, in software development, product objectives are sometimes tested by product user representatives. When the design is complete, coding follows and the finished code is then tested at the unit or module level by each programmer; at the component level by the group of programmers involved; and at the system level when all components are combined together. At early or late stages, a product or service may also be tested for usability.

At the system level, the manufacturer or independent reviewer may subject a product or service to one or more performance tests, possibly using one or more benchmarks. Whether viewed as a product or a service or both, a Web site can also be tested in various ways - by observing user experiences, by asking questions of users, by timing the flow through specific usage scenarios, and by comparing it with other sites.

For ASIC IP we have unit testing at the basic cell level. Module level testing is conducted at the EDA view level. System level testing is conducted at the ASIC level. See figure 1.

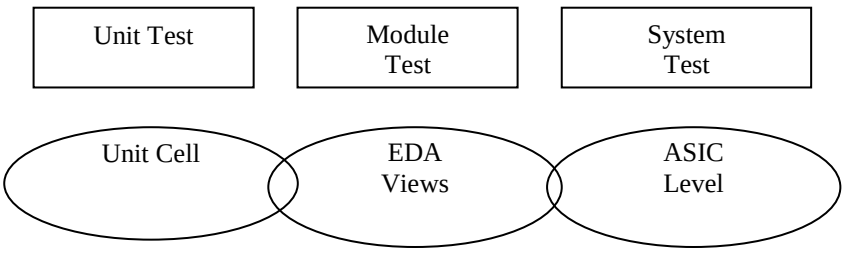


Figure 1. IP Testing related to Industry Standard Testing practices

The answer to the primary role of testing is two fold:

Determine whether the system meets specifications (Producer View), and Determine whether the system meets business and user needs (Customer View)

1.3. Goal of Testing

The primary Goal of Testing is to uncover requirement, design or coding errors in the programs or products. Let us look at some fundamental concepts of Testing.

"Verification" is the process of determining whether or not the products of a given phase of development fulfill the specifications established during the previous phase. The verification activities include proving, testing and reviews.

"Validation" is the process of evaluating the product at the end of the development to ensure compliance with requirements. Testing is a common method of validation. Validation testing, Compliance Testing and Conformance Testing terminology is used interchangeably throughout industry. They are inherently imply the same process.

For high reliability we need to perform both activities. Together they are called the "V & V" activities.
Testing is an Art. A good tester can always be a good developer, but a good developer need not be a good tester. In most cases developers struggle with testing.

1.4. Test Objectives

The following can be described as test objectives as per Glen Mayer:
Testing is a process of executing a program with the intent of finding an error.
A good test case is one that has a high probability of finding an as-yet-undiscovered error.
A successful test is one that uncovers an as-yet-undiscovered error.

1.5. Testing Principles

The following can be described as testing principles: All tests should be traceable to customer requirements.
Tests should be planned long before testing begins.
The Pareto principle applies to testing.
Testing should begin "in small" and progress toward testing "in large". (Incremental Testing Strategies)
Exhaustive testing is not possible.
To be most effective, testing should be conducted by an independent third party.

1.6. Verification Strategies

The following can be categorized into Verification Strategies: Requirements Reviews.
Design Reviews.
Code and Product Walkthrough.
Code and Product Inspection.

1.7. Validation Strategies

The following can be described as the basic Validation Test strategies.

- Unit Testing.
- Integration Testing.
- System Testing.
- Performance Testing.
- Alpha Testing.
- User Acceptance Testing (UAT)
- Installation Testing.
- Beta Testing.

1.8. Ok so now what is Regression Testing Anyhow?

The selective retesting of a product that has been modified to ensure that any bugs have been fixed and that no other previously working functions have failed as a result of the reparations and that newly added features have not created problems with previous versions of the product. Also referred to as *verification testing*, regression testing is initiated after a developer has attempted to fix a recognized problem or has added new features to a product that may have inadvertently introduced errors. It is a quality control measure to ensure that the new modifications still complies with its specified requirements and that unmodified portions of the product has not been affected by the maintenance activity. The intent of regression is to measure change over time and ensure that only the intended changes occur. Regression testing assumes that all initial validation of the product has been successfully completed and signed-off by the customer. Regression testing uses validation tests for verification of changes due to modifications or bug fixes.

1.9. Tools Used in Testing.

ASIC IP development Requirements definitions.
There are several layers to requirements docs that are needed to fully implement a QA testing strategy for Lib & IP Products.
To help understand the purpose and function we have to classify the types of requirements docs used in developing a solid

QA platform for Lib&IP.

1.10. **Product Specifications.**

Product Specifications provide detailed system level performance requirements and characteristics. Product specifications can also contain the conceptual requirements (Typically referred to as a conceptual specification). Often times conceptual specifications and product specifications are used interchangeably. An example of a product specification would be a library specification. It would identify process, speed, area, power, classes of cells, drive strength requirements, interoperability and compatibility requirements, tooling requirements, etc.

1.11. **Process Specifications.**

Process specifications provide guidelines for how all products will conform to generalized customer requirements. An example of a process specification would be a style manual, modeling guides, test strategy documents, etc. They provide a top level requirement that can be applied to any product going through the work flow. Example of the type of requirement would be pin orders on all EDA views. Regardless of the type of IP (memories, pads, cores, specialty IP) all product types must be in compliance to the process specifications and validated to them.

1.12. **Design Specifications.**

Design specifications: also referred to as detailed design specifications define details about the parts identified in the product specification. They contain details about unit interoperability with other products identified in the product specification, detailed behavior, detailed standards, and details about construction.

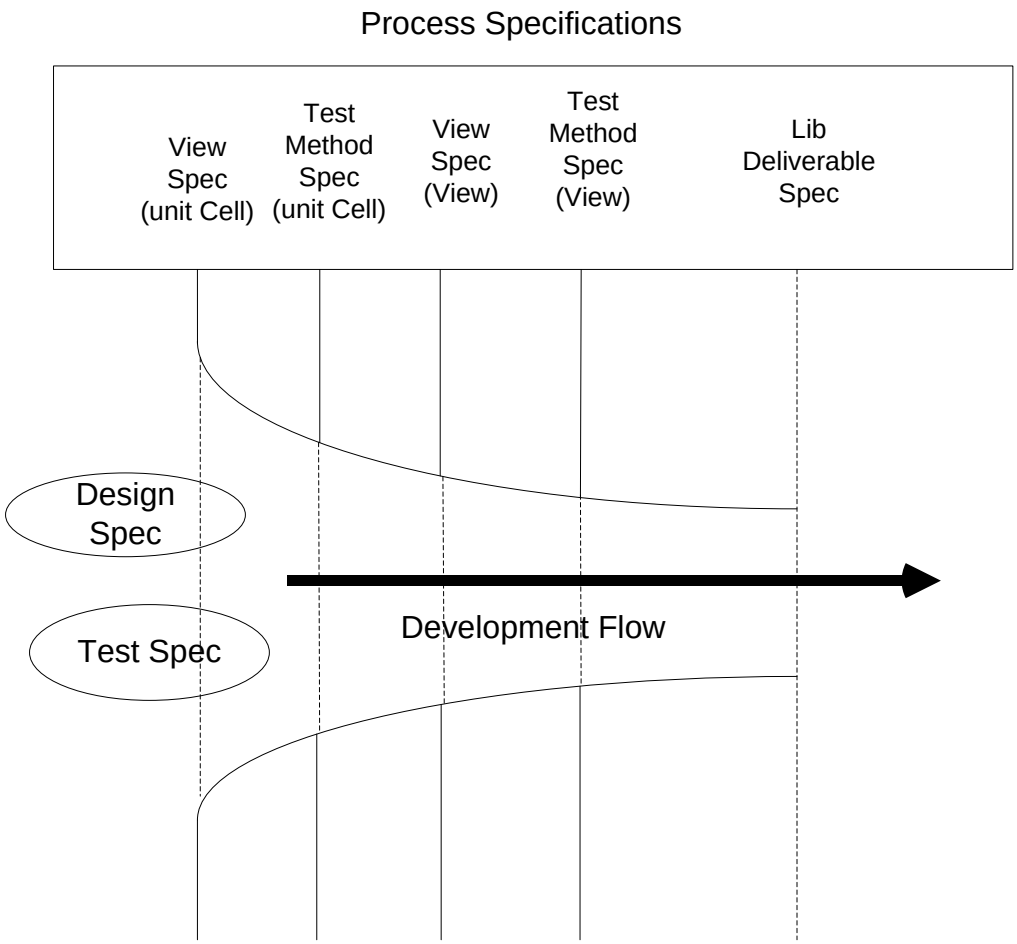
1.13. **Test Specifications.**

Test specifications have several layers to them. One level relates to process specification testing. The other level relates to design specification testing. Format checking typically applies to process specification testing. Behavioral validation testing relates to design specifications. Both must be successfully employed to ensure overall product quality is maintained and reduce rework.

1.14. **Release Specifications.**

Release specifications relate to delivery system requirements for products after develop is completed. Its after we have built it, how do you want us to give it to you.

Figure 3 is an example of how these requirements operate within the development flows.



2. TYPES OF TESTING PROCESS AUDITS

There can be various reasons to conduct Audits. The Audits may serve aim to achieve certain definite goals. Based on that we can classify them as follows:

Audit to verify compliance: In this type of auditing the prime motivation is to judge if the process complies with a standards. In these scenarios, the actual testing process is compared with the documented process. For example ISO Standards require us to define our IP testing process. The audit will try to verify if we actually conducted the testing as documented

Audit for process improvement/problem solving:

In this type of audit the motivation is to audit and trace the various steps in the process and try to weed out process problems. For instance it is observed that too many product defects escaped detection even though the testing process was apparently followed. So the audit is done as a preliminary step to collect facts and analyze them.

Audit for Root Cause Analysis

In this type of audit the motivation is to audit the testing process is to find a Root Cause of a specific problem. For example the customers discovered a huge problem with the IP. So we retrace our testing steps to find out what went wrong in this specific case.

Internal Audits

Typically the internal audits are initiated from within the organizations

External Audits

External Audits are done by and initiated by external agencies

3. WHY AUDIT THE TESTING PROCESS

Auditing Test Process helps the management understand if the process is being followed as specified. Typically Testing audit may be done for one or more of the following factors:

- To ensure continued reliability and integrity of the process
- To verify compliance of standards (ISO, CMM, etc)
- To solve process related problems
- To find the root cause of a specific problem
- To detect or prevent Fraud
- To improve the Testing process

Auditing of the Testing process may also be done if the IP Product is a mission critical one such as used for Medical Life Support Systems

This is done to prevent any loop holes or bugs in the Chip

4. HOW TO AUDIT

Typically the Audit of the Testing Process will include the following steps:

- reviewing the Testing process as documented in the Quality Manual. This helps the auditor understand the process as defined
- Reviewing the deliverable documents at each step
- Document reviewed include
  - Test Strategy
  - Test Plans
  - Test Cases
  - Test Logs
  - Defects Tracked
  - Test Coverage Matrix
  - any other relevant records

Each of the above document provides a certain level of traceability that the process was followed and the necessary steps were taken

- Interviewing the Project Team at various levels – PM, Coordinator, Tester

Interviewing the Project Team members gives an understanding of the thought process prevalent in those conducting the Testing Process.

This can provide valuable insights over an above what was actually documented

Independent agencies may verify the Test Processes.

5. WHAT CAN BE AUDITED

5.1. Whether the test process deliverables exist as specified

The only thing that can be really verified in an audit is that the process deliverables exist. The process deliverables are taken as a proof that the necessary steps were taken to do the testing. For example if Test Logs exist, we assume that testing was done and the Test Logs were created as a result of actual tests executed.

A separate exercise may be initiated to verify the authenticity of the Test Logs or other test deliverables

5.2. Whether test cases created covered all requirements/use cases

This analysis reveals if the test coverage was sufficient. It indicates that whether the testing team did the best to provide adequate amount of testing

5.3. Whether all Defects were fixed

The Status of all the Defects logged is checked to verify if all were fixed and verified

5.4. Whether there are any known bugs in the IP released

Sometimes all the defects may not be fixed, the IP may be released with known problems. Test Logs would indicate the actual results and evidence of any bugs being present.

5.5. Whether the levels of testing was effective enough

If Defects pass thru the various levels of testing undetected, it may reflect poorly on the effectiveness of the testing process

- What were the number of defects (Defect Leaks) that went by undetected in each phase
- Number of iterations of testing in each level
- Time taken to test each module/component
- This data may be used for process improvement
- Versions of IP actually tested

6. WHAT IS USER ACCEPTANCE TASTING

User Acceptance Testing is often the final step before rolling out the application.
Usually the end users who will be using the applications test the application before 'accepting' the application.
This type of testing gives the end users the confidence that the application being delivered to them meets their requirements.
This testing also helps nail bugs related to usability of the application.

7. USER ACCEPTANCE TESTING – PREREQUISITES

Before the User Acceptance testing can be done the IP product is fully developed.
Various levels of testing (Unit, Integration and System) are already completed before User Acceptance Testing is done. As various levels of testing have been completed most of the technical bugs have already been fixed before UAT.

8. USER ACCEPTANCE TESTING – WHAT TO TEST

To ensure an effective User Acceptance Testing Test cases are created.
These Test cases can be created using various use cases identified during the Requirements definition stage.
The Test cases ensure proper coverage of all the scenarios during testing.
During this type of testing the specific focus is the exact real world usage of the IP product. The Testing is done in an environment that simulates the production environment.
The Test cases are written using real world scenarios for the Product

9. USER ACCEPTANCE TESTING – HOW TO TEST

The user acceptance testing is usually a black box type of testing. In other words, the focus is on the functionality and the usability of the product rather than the technical aspects. It is generally assumed that the application would have already undergone Unit, Integration and System Level Testing.
However, it is useful if the User acceptance Testing is carried out in an environment that closely resembles the real world or production environment.
The steps taken for User Acceptance Testing typically involve one or more of the following:

- User Acceptance Test (UAT) Planning
- Designing UA Test Cases
- Selecting a Team that would execute the (UAT) Test Cases
- Executing Test Cases
- Documenting the Defects found during UAT
- Resolving the issues/Bug Fixing
- Sign Off

User Acceptance Test (UAT) Planning:
As always the Planning Process is the most important of all the steps. This affects the effectiveness of the Testing Process. The Planning process outlines the User Acceptance Testing Strategy. It also describes the key focus areas, entry and exit criteria.
Designing UA Test Cases:
The User Acceptance Test Cases help the Test Execution Team to test the application thoroughly. This also helps ensure that the UA Testing provides sufficient coverage of all the scenarios.
The Use Cases created during the Requirements definition phase may be used as inputs for creating Test Cases. The inputs from Business Analysts and Subject Matter Experts are also used for creating.

Each User Acceptance Test Case describes in a simple language the precise steps to be taken to test something.

The Business Analysts and the Project Team review the User Acceptance Test Cases.

Selecting a Team that would execute the (UAT) Test Cases:

Selecting a Team that would execute the UAT Test Cases is an important step.

The UAT Team is generally a good representation of the real world end users.

The Team thus comprises of the actual end users who will be using the application.

Executing Test Cases:

The Testing Team executes the Test Cases and may additional perform random Tests relevant to them

Documenting the Defects found during UAT:

The Team logs their comments and any defects or issues found during testing.

Resolving the issues/Bug Fixing:

The issues/defects found during Testing are discussed with the Project Team, Subject Matter Experts and Business Analysts. The issues are resolved as per the mutual consensus and to the satisfaction of the end users.

Sign Off:

Upon successful completion of the User Acceptance Testing and resolution of the issues the team generally indicates the acceptance of the application. This step is important in commercial IP sales or distribution. Once the User "Accept" the IP delivered they indicate that the IP meets their requirements.

The users now confident of the IP solution delivered and the vendor can be paid for the same.

10. UNIT TESTING: WHY? WHAT? & HOW?

In this tutorial you will learn about unit testing, various levels of testing, various types of testing based upon the intent of testing, How does Unit Testing fit into the IP product Life Cycle? Unit Testing Tasks and Steps, What is a Unit Test Plan? What is a Test Case? and Test Case Sample, Steps to Effective Unit Testing.

There are various levels of testing:

- Unit Testing
- Integration Testing
- System Testing

There are various types of testing based upon the intent of testing such as:

- Acceptance Testing
- Performance Testing
- Load Testing
- Regression Testing

Based on the testing Techniques testing can be classified as:

- Black box Testing

White box Testing

11. HOW DOES UNIT TESTING FIT INTO THE IP DEVELOPMENT LIFE CYCLE?

This is the first and the most important level of testing. As soon as the designer/developer develops a unit cell the unit cell is tested for various scenarios. As the product is built it is much more economical to find and eliminate the bugs early on. Hence Unit Testing is the most important of all the testing levels. As the IP project progresses ahead it becomes more and more costly to find and fix the bugs.

In most cases it is the developer's responsibility to deliver Unit Tested IP.

Unit Testing Tasks and Steps:

Step 1: Create a Test Plan

Step 2: Create Test Cases and Test Data

Step 3: If applicable create scripts to run test cases

Step 4: Once the IP is ready execute the test cases

Step 5: Fix the bugs if any and re test the IP

Step 6: Repeat the test cycle until the "unit cell" is free of all bugs

12. WHAT IS A UNIT TEST PLAN

This document describes the Test Plan in other words how the tests will be carried out.

This will typically include the list of things to be Tested, Roles and Responsibilities, prerequisites to begin Testing, Test Environment, Assumptions, what to do after a test is successfully carried out, what to do if test fails, Glossary and so on

13. WHAT IS A TEST CASE

Simply put, a Test Case describes exactly how the test should be carried out.

For example the test case may describe a test as follows:

Case 1: Memory Read during Write

Case 2: Valid Read

Test Cases clubbed together form a Test Suite

Test Case Sample

Test Case ID	Test Case Description	Input Data	Expected Result	Actual Result	Pass/Fail	Remarks

Additionally the following information may also be captured:

- Unit Cell Name and Version Being tested
- Tested By
- Date
- Test Iteration (One or more iterations of unit testing may be performed)

14. STEPS TO EFFECTIVE UNIT TESTING

1) Documentation: Early on document all the Test Cases needed to test your product. A lot of times this task is not given due importance. Document the Test Cases, actual Results when executing the Test Cases. There are several important advantages if the test cases and the actual execution of test cases are well documented.

- Documenting Test Cases prevents oversight.
- Documentation clearly indicates the quality of test cases
- If the IP needs to be retested we can be sure that we did not miss anything
- It provides a level of transparency of what was really tested during unit testing. This is one of the most important aspects.
- It helps in knowledge transfer in case of employee attrition
- Sometimes Unit Test Cases can be used to develop test cases for other levels of testing

2) What should be tested when Unit Testing: A lot depends on the type of IP or unit cell that is being created. It could be a memory or a pad or a core cell. Broadly the following aspects should be considered:

- Cell is properly declared.
- Structure of cell is in accordance with process specifications.
- Functionality meets design specifications.
- EDA tool compatibility
- Illegal functionality is properly protected or covered (also known and violation handling strategy)
- Cross compatibility with related and dependent views.
- Timing, area, power is consistent with product performance specifications and checked against actual working silicon.
- Create Test Cases such that every line of code in the unit is tested at least once in a test cycle
- Create Test Cases such that every condition in case of “conditional statements” is tested once

- Create Test Cases to test the minimum/maximum range of data that can be entered.
- Create Test Cases to verify how various errors are handled
- Create Test Cases to verify if all the validations are being performed

3) Automate where Necessary: Time pressures/Pressure to get the job done may result in developers cutting corners in unit testing. Sometimes it helps to write scripts, which automate a part of unit testing. This may help ensure that the necessary tests were done and may result in saving time required to perform the tests.

15. INTEGRATION TESTING: WHY? WHAT? & HOW?

Introduction:

As we covered in various articles in the Testing series there are various levels of testing:
Unit Testing, Integration Testing, System Testing
Each level of testing builds on the previous level.
"Unit testing" focuses on testing a unit of the code.
"Integration testing" is the next level of testing. This 'level of testing' focuses on testing the integration of "units of code" or components.

16. HOW DOES INTEGRATION TESTING FIT INTO THE IP DEVELOPMENT LIFE CYCLE

Even if a component is successfully unit tested, in a library of components it is of little or no value if the component cannot be successfully integrated with the rest of the components and IP.

Once unit tested components are delivered we then integrate them together. These "integrated" components are tested to weed out errors and bugs caused due to the integration. This is a very important step in the IP Development life Cycle.

It is possible that different developers developed different components in different ways.
A lot of bugs emerge during the integration step.
In most cases a dedicated testing team focuses on Integration Testing.

17. PREREQUISITES FOR INTEGRATION TESTING

Before we begin Integration Testing it is important that all the components have been successfully unit tested.

18. INTEGRATION TESTING STEPS

Integration Testing typically involves the following Steps:
Step 1: Create a Test Plan
Step 2: Create Test Cases and Test Data
Step 3: If applicable create scripts to run test cases
Step 4: Once the components have been integrated execute the test cases
Step 5: Fix the bugs if any and re test the IP
Step 6: Repeat the test cycle until the components have been successfully integrated

19. WHAT IS AN "INTEGRATION TEST PLAN"?

As you may have read in the other articles in the series, this document typically describes one or more of the following:

- How the tests will be carried out
- The list of things to be Tested
- Roles and Responsibilities
- Prerequisites to begin Testing
- Test Environment
- Assumptions
- What to do after a test is successfully carried out

- What to do if test fails
- Glossary

20. HOW TO WRITE AN INTEGRATION TEST CASE?

Simply put, a Test Case describes exactly how the test should be carried out.
The Integration test cases specifically focus on the flow of data/information/control from one component to the other.

So the Integration Test cases should typically focus on scenarios where one component is being called from another. Also the overall IP functionality should be tested to make sure the components works when the different components are brought together.

The various Integration Test Cases clubbed together form an Integration Test Suite
Each suite may have a particular focus. In other words different Test Suites may be created to focus on different areas of the components in the IP.

As mentioned before a dedicated Testing Team may be created to execute the Integration test cases. Therefore the Integration Test Cases should be as detailed as possible.

Sample Test Case Table:

Test Case ID	Test Case Description	Input Data	Expected Result	Actual Result	Pass/Fail	Remarks

Additionally the following information may also be captured:
a) Test Suite Name
b) Tested By
c) Date
d) Test Iteration (One or more iterations of Integration testing may be performed)

21. WORKING TOWARDS EFFECTIVE INTEGRATION TESTING

There are various factors that affect Integration and hence Integration Testing:

1) Configuration Management: Since Integration Testing focuses on Integration of components and components can be built by different developers and even different development teams, it is important the right version of components are tested. This may sound very basic, but the biggest problem faced in development is integrating the right version of components. Integration testing may run through several iterations and to fix bugs components may undergo changes. Hence it is important that a good Configuration Management (CM) policy is in place. We should be able to track the components and their versions. So each time we integrate the application components we know exactly what versions go into the build process.

2) Automate Build Process where Necessary: A Lot of errors occur because the wrong version of components were sent for the build or there are missing components. If possible write a script to integrate and deploy the components this helps reduce manual errors.

3) Document: Document the Integration process/build process to help eliminate the errors of omission or oversight. It is possible that the person responsible for integrating the components forgets to run a required script and the Integration Testing will not yield correct results.

4) Defect Tracking: Integration Testing will lose its edge if the defects are not tracked correctly. Each defect should be documented and tracked. Information should be captured as to how the defect was fixed. This is valuable information. It can help in future integration and deployment processes.

22. SYSTEM TESTING: WHY? WHAT? & HOW?

Introduction:

'**Unit testing**' focuses on testing each unit of the code.

'**Integration testing**' focuses on testing the integration of "units of code" or components.
Each level of testing builds on the previous level.

'System Testing' is the next level of testing. It focuses on testing the IP products as a whole.

This article attempts to take a close look at the System Testing Process and analyze:
Why System Testing is done? What are the necessary steps to perform System Testing? How to make it successful?

23. HOW DOES SYSTEM TESTING FIT INTO THE IP DEVELOPMENT LIFE CYCLE?

In a typical Enterprise, 'unit testing' is done by the developers. This ensures that the individual components are working OK. The 'Integration testing' focuses on successful integration of all the individual pieces of components or units of code.

Once the components are integrated, the system as a whole needs to be rigorously tested to ensure that it meets the Quality Standards.

Thus the System testing builds on the previous levels of testing namely unit testing and Integration Testing.

Usually a dedicated testing team is responsible for doing 'System Testing'.

24. WHY SYSTEM TESTING IS IMPORTANT?

System Testing is a crucial step in Quality Management Process.

- In the IP Development life cycle System Testing is the first level where the System is tested as a whole
- The System is tested to verify if it meets the functional and technical requirements
- The application/System is tested in an environment that closely resembles the production environment where the application will be finally deployed
- The System Testing enables us to test, verify and validate both the Business requirements as well as the IP Architecture

25. PREREQUISITES FOR SYSTEM TESTING:

The prerequisites for System Testing are:

- All the components should have been successfully Unit Tested
- All the components should have been successfully integrated and Integration Testing should be completed
- An Environment closely resembling the production environment should be created

When necessary, several iterations of System Testing are done in multiple environments.

26. STEPS NEEDED TO DO SYSTEM TESTING:

The following steps are important to perform System Testing:
Step 1: Create a System Test Plan
Step 2: Create Test Cases
Step 3: Carefully Build Data used as Input for System Testing
Step 3: If applicable create scripts to

- Build environment and
- to automate Execution of test cases

Step 4: Execute the test cases
Step 5: Fix the bugs if any and re test
Step 6: Repeat the test cycle as necessary

27. WHAT IS A "SYSTEM TEST PLAN"?

As you may have read in the other articles in the testing series, this document typically describes the following:

- The Testing Goals
- The key areas to be focused on while testing
- The Testing Deliverables
- How the tests will be carried out
- The list of things to be Tested

- Roles and Responsibilities
- Prerequisites to begin Testing
- Test Environment
- Assumptions
- What to do after a test is successfully carried out
- What to do if test fails
- Glossary

28. HOW TO WRITE A SYSTEM TEST CASE?

A Test Case describes exactly how the test should be carried out.

The System test cases help us verify and validate the system.

The System Test Cases are written such that:

- They cover all the use cases and scenarios
- The Test cases validate the technical Requirements and Specifications
- The Test cases verify if the application/System meet the Business & Functional Requirements specified
- The Test cases may also verify if the System meets the performance standards

Since a dedicated test team may execute the test cases it is necessary that System Test Cases. The detailed Test cases help the test executioners do the testing as specified without any ambiguity.

The format of the System Test Cases may be like all other Test cases as illustrated below:

- Test Case ID
- Test Case Description:
  - What to Test?
  - How to Test?
- Input Data
- Expected Result
- Actual Result

Sample Test Case Format:

Test Case ID	What To Test?	How to Test?	Input Data	Expected Result	Actual Result	Pass/Fail
.	.	.	.	.	.	.

Additionally the following information may also be captured:

a) Test Suite Name

b) Tested By

c) Date

d) Test Iteration (The Test Cases may be executed one or more times)

29. WORKING TOWARDS EFFECTIVE SYSTEMS TESTING

There are various factors that affect success of System Testing:

1) Test Coverage: System Testing will be effective only to the extent of the coverage of Test Cases. What is Test coverage? Adequate Test coverage implies the scenarios covered by the test cases are sufficient. The Test cases should “cover” all scenarios, use cases, Business Requirements, Technical Requirements, and Performance Requirements. The test cases should enable us to verify and validate that the system/application meets the project goals and specifications.

2) Defect Tracking: The defects found during the process of testing should be tracked. Subsequent iterations of test cases verify if the defects have been fixed.

3) Test Execution: The Test cases should be executed in the manner specified. Failure to do so results in improper Test Results.

4) Build Process Automation: A Lot of errors occur due to an improper build. ‘Build’ is a compilation of the various components that make the application deployed in the appropriate environment. The Test results will not be accurate if the

application is not 'built' correctly or if the environment is not set up as specified. Automating this process may help reduce manual errors.

5) Test Automation: Automating the Test process could help us in many ways:

a. The test can be repeated with fewer errors of omission or oversight

b. Some scenarios can be simulated if the tests are automated for instance simulating a large number of gates or simulating increasing large amounts of input/output data

6) Documentation: Proper Documentation helps keep track of Tests executed. It also helps create a knowledge base for current and future projects. Appropriate metrics/Statistics can be captured to validate or verify the efficiency of the technical design /architecture.

30. WHAT IS REGRESSION TESTING?

If a piece of IP is modified for any reason testing needs to be done to ensure that it works as specified and that it has not negatively impacted any functionality that it offered previously. This is known as Regression Testing.

Regression Testing attempts to verify:

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

31. WHEN IS REGRESSION TESTING NECESSARY?

Regression Testing plays an important role in any Scenario where a change has been made to a previously tested product. Regression Testing is hence an important aspect in various Design Flow Methodologies where IP changes and enhancements occur frequently.

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

Some Development Methodologies embrace change. Some don't.

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

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

32. WHY IS REGRESSION TESTING IMPORTANT?

Any IP change can cause existing functionality to break.
Changes to a component could impact dependent Components.

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

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

33. MAKING REGRESSION TESTING COST EFFECTIVE:

Every time a change occurs one or more of the following scenarios may occur:

- More Functionality may be added to the system
- More complexity may be added to the system
- New bugs may be introduced
- New vulnerabilities may be introduced in the system
- System may tend to become more and more fragile with each change

After the change the new functionality may have to be tested along with all the original functionality.

With each change Regression Testing could become more and more costly.

To make the Regression Testing Cost Effective and yet ensure good coverage one or more of the following techniques may be applied:

- Test Automation: If the Test cases are automated the test cases may be executed using scripts after each change is introduced in the system. The execution of test cases in this way helps eliminate oversight, human errors,. It may also result in faster and cheaper execution of Test cases. However there is cost involved in building the scripts.

- Selective Testing: Some Teams choose execute the test cases selectively. They do not execute all the Test Cases during the Regression Testing. They test only what they decide is relevant. This helps reduce the Testing Time and Effort.

34. REGRESSION TESTING – WHAT TO TEST?

Since Regression Testing tends to verify the IP after a change has been made everything that may be impacted by the change should be tested during Regression Testing. Generally the following areas are covered during Regression Testing:

- Any functionality that was addressed by the change
- Original Functionality of the system
- Performance of the System after the change was introduced

35. REGRESSION TESTING – HOW TO TEST?

Like any other Testing Regression Testing Needs proper planning.
For an Effective Regression Testing to be done the following ingredients are necessary:

- Create a Regression Test Plan: Test Plan identified Focus Areas, Strategy, Test Entry and Exit Criteria. It can also outline Testing Prerequisites, Responsibilities, etc.

- Create Test Cases: Test Cases that cover all the necessary areas are important. They describe what to Test, Steps needed to test, Inputs and Expected Outputs. Test Cases used for Regression Testing should specifically cover the functionality addressed by the change and all components affected by the change. The Regression Test case may also include the testing of the performance of the components and the application after the change(s) were done.

- Defect Tracking: As in all other Testing Levels and Types It is important Defects are tracked systematically, otherwise it undermines the Testing Effort.

36. METRICS USED IN TESTING

In this tutorial you will learn about metrics used in testing, The Product Quality Measures - 1. Customer satisfaction index, 2. Delivered defect quantities, 3. Responsiveness (turnaround time) to users, 4. Product volatility, 5. Defect ratios, 6. Defect removal efficiency, 7. Complexity of delivered product, 8. Test coverage, 9. Cost of defects, 10. Costs of quality activities, 11. Re-work, 12. Reliability and Metrics for Evaluating Application System Testing.

37. THE PRODUCT QUALITY MEASURES:

37.1. Customer satisfaction index

This index is surveyed before product delivery and after product delivery (and on-going on a periodic basis, using standard questionnaires).The following are analyzed:

- Number of system enhancement requests per year
- Number of maintenance fix requests per year
- User friendliness: call volume to customer service hotline
- User friendliness: training time per new user
- Number of product recalls or fix releases (software vendors)
- Number of production re-runs (in-house information systems groups)

37.2. Delivered defect quantities (Lines of Code (LOC), Number of Gates (NOG))

They are normalized per function point (or per LOC or NOG) at product delivery (first 3 months or first year of operation) or Ongoing (per year of operation) by level of severity, by category or cause, e.g.: requirements defect, design defect, code defect, documentation/on-line help defect, defect introduced by fixes, etc.

37.3. Responsiveness (turnaround time) to users

- Turnaround time for defect fixes, by level of severity
- Time for minor vs. major enhancements; actual vs. planned elapsed time

37.4. Product volatility

Ratio of maintenance fixes (to repair the system & bring it into compliance with specifications), vs. enhancement requests (requests by users to enhance or change functionality)

37.5. Defect ratios

- Defects found after product delivery per function point.
- Defects found after product delivery per LOC or NOG
- Pre-delivery defects: annual post-delivery defects
- Defects per function point of the system modifications

37.6. Defect removal efficiency

- Number of post-release defects (found by clients in field operation), categorized by level of severity
- Ratio of defects found internally prior to release (via inspections and testing), as a percentage of all defects
- All defects include defects found internally plus externally (by customers) in the first year after product delivery

37.7. Complexity of delivered product

- McCabe's cyclomatic complexity counts across the system
- Halstead's measure
- Card's design complexity measures
- Predicted defects and maintenance costs, based on complexity measures

37.8. Test coverage

- Breadth of functional coverage
- Percentage of paths, branches or conditions that were actually tested
- Percentage by criticality level: perceived level of risk of paths
- The ratio of the number of detected faults to the number of predicted faults.

37.9. Cost of defects

- Business losses per defect that occurs during operation
- Business interruption costs; costs of work-arounds
- Lost sales and lost goodwill
- Litigation costs resulting from defects
- Annual maintenance cost (per function point)
- Annual operating cost (per function point)
- Measurable damage to your boss's career

37.10. Costs of quality activities

- Costs of reviews, inspections and preventive measures
- Costs of test planning and preparation
- Costs of test execution, defect tracking, version and change control
- Costs of diagnostics, debugging and fixing
- Costs of tools and tool support
- Costs of test case library maintenance
- Costs of testing & QA education associated with the product
- Costs of monitoring and oversight by the QA organization (if separate from the development and test organizations)

37.11. Re-work

- Re-work effort (hours, as a percentage of the original product hours)
- Re-worked LOC (source lines of code, as a percentage of the total delivered LOC) or NOG (number of gates in a component to the total number of gates used in the library deliverable)
- Re-worked components (as a percentage of the total delivered components)

37.12. Reliability

- Availability (percentage of time a system is available, versus the time the system is needed to be available)
- Mean time between failure (MTBF).
- Man time to repair (MTTR)

- Reliability ratio (MTBF / MTTR)
- Number of product recalls or fix releases
- Number of production re-runs as a ratio of production runs

38. METRICS FOR EVALUATING APPLICATION SYSTEM TESTING:

Metric = Formula
Test Coverage = Number of units (KLOC/FP) tested / total size of the system. (LOC represents Lines of Code)
Number of tests per unit size = Number of test cases per KLOC/FP (LOC represents Lines of Code).
Acceptance criteria tested = Acceptance criteria tested / total acceptance criteria
Defects per size = Defects detected / system size
Test cost (in %) = Cost of testing / total cost *100
Cost to locate defect = Cost of testing / the number of defects located
Achieving Budget = Actual cost of testing / Budgeted cost of testing
Defects detected in testing = Defects detected in testing / total system defects
Defects detected in production = Defects detected in production/system size
Quality of Testing = No of defects found during Testing/(No of defects found during testing + No of acceptance defects found after delivery) *100
Effectiveness of testing to business = Loss due to problems / total resources processed by the system.
System complaints = Number of third party complaints / number of transactions processed
Scale of Ten = Assessment of testing by giving rating in scale of 1 to 10
Source Code Analysis = Number of source code statements changed / total number of tests.
Effort Productivity = Test Planning Productivity = No of Test cases designed / Actual Effort for Design and Documentation
Test Execution Productivity = No of Test cycles executed / Actual Effort for testing

39. TECHNICAL TERMS USED IN TESTING WORLD

In this tutorial you will learn about technical terms used in testing world, from Audit, Acceptance Testng to Validation, Verification and Testing.

Audit: An independent examination of a work product or set of work products to assess compliance with specifications, standards, contractual agreements, or other criteria.

Acceptance testing: Testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.

Alpha Testing: Acceptance testing performed by the customer in a controlled environment at the developer's site. The software is used by the customer in a setting approximating the target environment with the developer observing and recording errors and usage problems.

Assertion Testing: A dynamic analysis technique which inserts assertions about the relationship between program variables into the program code. The truth of the assertions is determined as the program executes.

Boundary Value: (1) A data value that corresponds to a minimum or maximum input, internal, or output value specified for a system or component. (2) A value which lies at, or just inside or just outside a specified range of valid input and output values.

Boundary Value Analysis: A selection technique in which test data are chosen to lie along "boundaries" of the input domain [or output range] classes, data structures, procedure parameters, etc. Choices often include maximum, minimum, and trivial values or parameters.

Branch Coverage: A test coverage criteria which requires that for each decision point each possible branch be executed at least once.

Bug: A fault in a program which causes the program to perform in an unintended or unanticipated manner.

Beta Testing: Acceptance testing performed by the customer in a live application of the software, at one or more end user sites, in an environment not controlled by the developer.

Boundary Value Testing: A testing technique using input values at, just below, and just above, the defined limits of an input domain; and with input values causing outputs to be at, just below, and just above, the defined limits of an output domain.

Branch Testing: Testing technique to satisfy coverage criteria which require that for each decision point, each possible branch [outcome] be executed at least once. Contrast with testing, path; testing, statement. See: branch coverage.

Compatibility Testing: The process of determining the ability of two or more systems to exchange information. In a situation where the developed software replaces an already working program, an investigation should be conducted to assess possible comparability problems between the new software and other programs or systems.

Cause Effect Graph: A Boolean graph linking causes and effects. The graph is actually a digital-logic circuit (a combinatorial logic network) using a simpler notation than standard electronics notation.

Cause Effect Graphing: This is a Test data selection technique. The input and output domains are partitioned into classes and analysis is performed to determine which input classes cause which effect. A minimal set of inputs is chosen which will cover the entire effect set. It is a systematic method of generating test cases representing combinations of conditions.

Code Inspection: A manual [formal] testing [error detection] technique where the programmer reads source code, statement by statement, to a group who ask questions analyzing the program logic, analyzing the code with respect to a checklist of historically common programming errors, and analyzing its compliance with coding standards.

Code Review: A meeting at which software code is presented to project personnel, managers, users, customers, or other interested parties for comment or approval.

Code Walkthrough: A manual testing [error detection] technique where program [source code] logic [structure] is traced manually [mentally] by a group with a small set of test cases, while the state of program variables is manually monitored, to analyze the programmer's logic and assumptions.

Coverage Analysis: Determining and assessing measures associated with the invocation of program structural elements to determine the adequacy of a test run. Coverage analysis is useful when attempting to execute each statement, branch, path, or iterative structure in a program.

Crash: The sudden and complete failure of a computer system or component.

Criticality: The degree of impact that a requirement, module, error, fault, failure, or other item has on the development or operation of a system.

Cyclomatic Complexity: The number of independent paths through a program. The cyclomatic complexity of a program is equivalent to the number of decision statements plus 1.

Error: A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

Error Guessing: This is a Test data selection technique. The selection criterion is to pick values that seem likely to cause errors.

Error Seeding: The process of intentionally adding known faults to those already in a computer program for the purpose of monitoring the rate of detection and removal, and estimating the number of faults remaining in the program. Contrast with mutation analysis.

Exception: An event that causes suspension of normal program execution. Types include addressing exception, data exception, operation exception, overflow exception, protection exception, and underflow exception.

Exhaustive Testing: Executing the program with all possible combinations of values for program variables. This type of testing is feasible only for small, simple programs.

Failure: The inability of a system or component to perform its required functions within specified performance requirements.

Fault: An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner.

Functional Testing: Testing that ignores the internal mechanism or structure of a system or component and focuses on the outputs generated in response to selected inputs and execution conditions. (2) Testing conducted to evaluate the compliance of a system or component with specified functional requirements and corresponding predicted results.

Integration Testing: An orderly progression of testing in which software elements, hardware elements, or both are combined and tested, to evaluate their interactions, until the entire system has been integrated.

Interface Testing: Testing conducted to evaluate whether systems or components pass data and control correctly to one another.

Mutation Testing: A testing methodology in which two or more program mutations are executed using the same test cases to evaluate the ability of the test cases to detect differences in the mutations.

Operational Testing: Testing conducted to evaluate a system or component in its operational environment.

Parallel Testing: Testing a new or an altered data processing system with the same source data that is used in another system. The other system is considered as the standard of comparison.

Path Testing: Testing to satisfy coverage criteria that each logical path through the program be tested. Often paths through the program are grouped into a finite set of classes. One path from each class is then tested.

Performance Testing: Functional testing conducted to evaluate the compliance of a system or component with specified performance requirements.

Qualification Testing: Formal testing, usually conducted by the developer for the consumer, to demonstrate that the software meets its specified requirements.

Quality Assurance: (1) The planned systematic activities necessary to ensure that a component, module, or system conforms to established technical requirements. (2) All actions that are taken to ensure that a development organization delivers products that meet performance requirements and adhere to standards and procedures. (3) The policy, procedures, and systematic actions established in an enterprise for the purpose of providing and maintaining some degree of confidence in data integrity and accuracy throughout the life cycle of the data, which includes input, update, manipulation, and output. (4) The actions, planned and performed, to provide confidence that all systems and components that influence the quality of the product are working as expected individually and collectively.

Quality Control: The operational techniques and procedures used to achieve quality requirements.

Regression Testing: Rerunning test cases which a program has previously executed correctly in order to detect errors spawned by changes or corrections made during software development and maintenance.

Review: A process or meeting during which a work product or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. Types include code review, design review, formal qualification review, requirements review, test readiness review.

Risk: A measure of the probability and severity of undesired effects.

Risk Assessment: A comprehensive evaluation of the risk and its associated impact.

Software Review: An evaluation of software elements to ascertain discrepancies from planned results and to recommend improvement. This evaluation follows a formal process. Syn: software audit. See: code audit, code inspection, code review, code walkthrough, design review, specification analysis, static analysis

Static Analysis: Analysis of a program that is performed without executing the program. The process of evaluating a system or component based on its form, structure, content, documentation is also called as Static Analysis.

Statement Testing: Testing to satisfy the criterion that each statement in a program be executed at least once during program testing.

Storage Testing: This is a determination of whether or not certain processing conditions use more storage [memory] than estimated.

Stress Testing: Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements.

Structural Testing: Testing that takes into account the internal mechanism [structure] of a system or component. Types include branch testing, path testing, statement testing. (2) Testing to insure each program statement is made to execute during testing and that each program statement performs its intended function.

System Testing: The process of testing an integrated hardware and software system to verify that the system meets its specified requirements. Such testing may be conducted in both the development environment and the target environment.

Test: An activity in which a system or component is executed under specified conditions, the results are observed or recorded and an evaluation is made of some aspect of the system or component.

Testability: The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

Testcase: Documentation specifying inputs, predicted results, and a set of execution conditions for a test item.

Testcase Generator: A software tool that accepts as input source code, test criteria, specifications, or data structure definitions; uses these inputs to generate test input data; and, sometimes, determines expected results.

Test Design: Documentation specifying the details of the test approach for a software feature or combination of software features and identifying the associated tests.

Test Documentation: Documentation describing plans for, or results of, the testing of a system or component, Types include test case specification, test incident report, test log, test plan, test procedure, test report.

Test Driver: A software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results.

Test Incident Report: A document reporting on any event that occurs during testing that requires further investigation.

Test Item: A software item which is the object of testing.

Test Log: A chronological record of all relevant details about the execution of a test.

Test Phase: The period of time in the software life cycle in which the components of a software product are evaluated and integrated, and the software product is evaluated to determine whether or not requirements have been satisfied.

Test Plan: Documentation specifying the scope, approach, resources, and schedule of intended testing activities. It identifies test items, the features to be tested, the testing tasks, responsibilities, required, resources, and any risks requiring contingency planning. See: test design, validation protocol.

Test Procedure: A formal document developed from a test plan that presents detailed instructions for the setup, operation, and evaluation of the results for each defined test.

Test Report: A document describing the conduct and results of the testing carried out for a system or system component.

Test Result Analyzer: A software tool used to test output data reduction, formatting, and printing.

Testing: (1) The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component. (2) The process of analyzing a software item to detect the differences between existing and required conditions, i.e. bugs, and to evaluate the features of the software items.

Traceability Matrix: A matrix that records the relationship between two or more products; e.g., a matrix that records the relationship between the requirements and the design of a given software component. See: traceability, traceability analysis.

Unit Testing: Testing of a module for typographic, syntactic, and logical errors, for correct implementation of its design, and for satisfaction of its requirements (or) Testing conducted to verify the implementation of the design for one software element; e.g., a unit or module; or a collection of software elements.

Usability: The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.

Usability Testing: Tests designed to evaluate the machine/user interface.

Validation: Establishing documented evidence which provides a high degree of assurance that a specific process will consistently produce a product meeting its predetermined specifications and quality attributes.

Validation, Verification and Testing: Used as an entity to define a procedure of review, analysis, and testing throughout the software life cycle to discover errors, determine functionality, and ensure the production of quality software.

Volume Testing: Testing designed to challenge a system's ability to manage the maximum amount of data over a period of time. This type of testing also evaluates a system's ability to handle overload situations in an orderly fashion.

40. ADVANTAGES OF AUTOMATED TESTING

40.1. Introduction:

"Automated Testing" is automating the manual testing process currently in use. This requires that a formalized "manual testing process", currently exists in the company or organization.

Automation is the use of strategies, tools and artifacts that augment or reduce the need of manual or human involvement or interaction in unskilled, repetitive or redundant tasks.

Minimally, such a process includes:

- Detailed test cases, including predictable "expected results", which have been developed from Business Functional Specifications and Design documentation
- A standalone Test Environment, including a Test Database that is restorable to a known constant, such that the test cases are able to be repeated each time there are modifications made to the application.

40.2. The following types of testing can be automated

- **Functional** - testing that operations perform as expected.
- **Regression** - testing that the behavior of the system has not changed.
- **Exception or Negative** - forcing error conditions in the system.
- **Stress** - determining the absolute capacities of the application and operational infrastructure.
- **Performance** - providing assurance that the performance of the system will be adequate for both batch runs and online transactions in relation to business projections and requirements.
- **Load** - determining the points at which the capacity and performance of the system become degraded to the situation that hardware or software upgrades would be required.

40.3. Benefits of Automated Testing

Reliable: Tests perform precisely the same operations each time they are run, thereby eliminating human error
Repeatable: You can test how the IP reacts under repeated execution of the same operations.
Programmable: You can program sophisticated tests that bring out hidden information from the design architecture.
Comprehensive: You can build a suite of tests that covers every feature in your IP.
Reusable: You can reuse tests on different versions of an application.
Better Quality IP: Because you can run more tests in less time with fewer resources
Fast: Automated Tools run tests significantly faster than human users.
Cost Reduction: As the number of resources for regression test are reduced.
Choosing the right tools for the job and targeting the right areas of the organization to deploy them can only realize these benefits. The right areas where the automation fit must be chosen.

40.4. The following areas must be automated first

- Highly redundant tasks or scenarios
- Repetitive tasks that are boring or tend to cause human error
- Well-developed and well-understood use cases or scenarios first
- Relatively stable areas of the application over volatile ones must be automated.

40.5. Automated testers must follow the following guidelines to get the benefits of automation:

- **Concise:** As simple as possible and no simpler.
- **Self-Checking:** Test reports its own results; needs no human interpretation.
- **Repeatable:** Test can be run many times in a row without human intervention.
- **Robust:** Test produces same result now and forever. Tests are not affected by changes in the external environment.
- **Sufficient:** Tests verify all the requirements of the IP being tested.
- **Necessary:** Everything in each test contributes to the specification of desired behavior.
- **Clear:** Every statement is easy to understand.
- **Efficient:** Tests run in a reasonable amount of time.
- **Specific:** Each test failure points to a specific piece of broken functionality; unit test failures provide "defect triangulation".
- **Independent:** Each test can be run by itself or in a suite with an arbitrary set of other tests in any order.
- **Maintainable:** Tests should be easy to understand and modify and extend.
- **Traceable:** To and from the IP it tests and to and from the requirements.

41. DISADVANTAGES OF AUTOMATION TESTING

Though the automation testing has many advantages, it has its own disadvantages too. Some of the disadvantages are:

- Proficiency is required to write the automation test scripts.
- Debugging the test script is major issue. If any error is present in the test script, sometimes it may lead to deadly consequences.
- Test maintenance is costly in case of playback methods. Even though a minor change occurs in the IP, the test script has to be rerecorded or replaced by a new test script.
- Maintenance of test data files is difficult, if the test script tests more IP.

Some of the above disadvantages often cause damage to the benefit gained from the automated scripts. Though the automation testing has pros and cons, it is adapted widely all over the world.