Manual Testing

International Software Testing Qualifications Board Certification Preparation

Introduction to Software Testing

Computer Bug

- In 1947 Harvard University was operating a room-sized computer called the Mark II.
- A moth flew into the computer and was zapped by the high voltage when it landed on a relay.
- Hence, the

Defect a.k.a.

- Bug
- Fault
- Problem
- Failure
- Error
- Incident

Categories of Defects

- **Functional** – Wrong, Missing and Extra:
  - The software does not do something that the specification says it should do
  - The software does something that the specification says it should not do
  - The software does something that the specification does not mention
  - The software does not do something that the product specification does not mention but should do
- **Non – Functional** – The software is difficult to understand, hard to use, slow

Cost of finding and fixing defects

- Bugs found later cost more to fix
- Cost to fix a bug increases exponentially (10^x)
- E.g., a bug found during specification costs $1 to fix
  - … if found in design cost is $10
  - … if found in code cost is $100
  - … if found in released software cost is $1000

Objective of Testing

- Find defects.

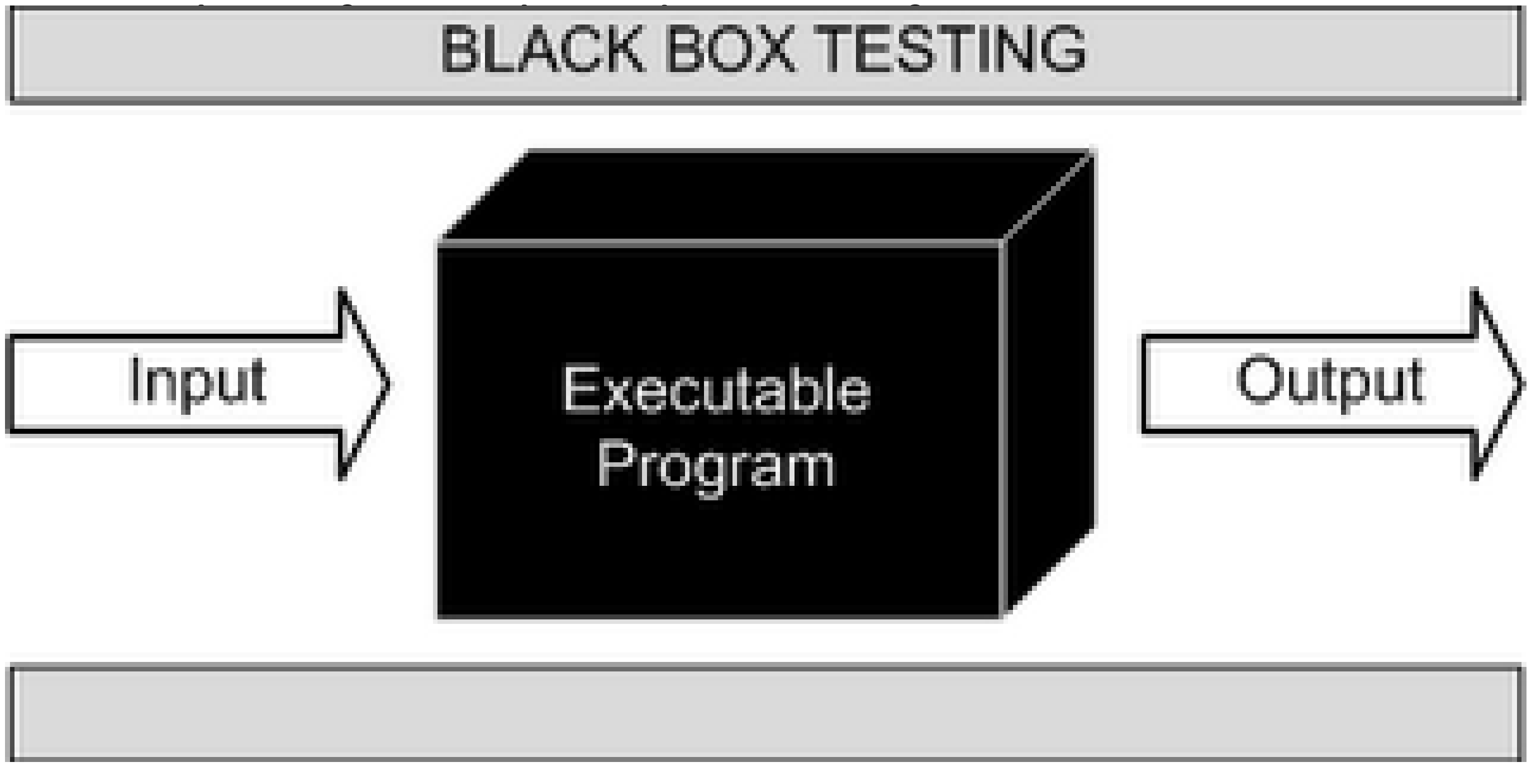
- GET them fixed from developers.

Software Testing Life Cycle

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

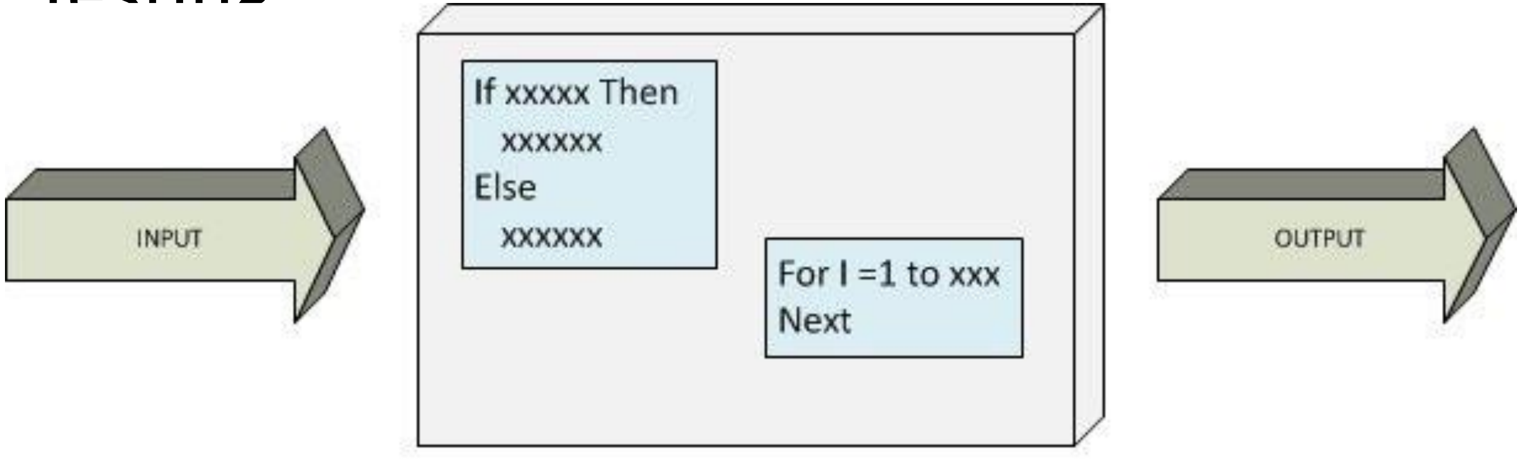
Test Method

- Black Box Testing / Functional Testing



Test Method

- White Box Testing / Glass Box / Structural Testing



White Box Testing

- Also called as Glass Box / Structural Testing.
- It is done by Programmers.
- This method tests internal structure or working of an application function / functionality.
- The tester (programmer) chooses inputs to exercise paths through the code and determine the appropriate outputs.

Static Techniques and Test Process

Static Techniques and Test Process

- Benefits of reviews include early defect detection and correction, development productivity improvements, reduced development timescales, reduced testing cost and time, lifetime cost reductions, fewer defects and improved communication. Reviews can find omissions, for example, in requirements, which are unlikely to be found in dynamic testing.

Static Techniques and Test Process

- Typical defects that are easier to find in reviews than in dynamic testing are: deviations from standards, requirement defects, design defects, insufficient maintainability and incorrect interface specifications.

Review Process

- The way a review is carried out depends on the agreed objective of the review (e.g. find defects, gain understanding, or discussion and decision by consensus).

Activities of a Formal Review

1. Planning
   - Defining the review criteria.
   - Selecting the personnel.
   - Allocating roles.
   - Defining the entry and exit criteria for more formal review types (e.g. inspections)
   - Selecting which part of the document is to be reviewed.
   - Checking the entry criteria (for more formal review types).

Activities of a Formal Review

2. Kick Off:
    - Distributing documents;
    - Explaining the objectives, process and documents to the participants.
3. Individual preparation:
    - Preparing for the review meeting by reviewing the document(s)
    - Noting potential defects, questions and comments.

Activities of a Formal Review

4. Examination/evaluation/recording of results (review meeting):
    - Discussion or logging, with documented results or minutes (for more formal review types).
    - Noting defects, make recommendations for handling the defects, make decisions about the defects.
    - Examining/evaluating and recording during any physical meetings or tracking any group electronic communications.

Activities of a Formal Review

5. Rework:
   - Fixing defects found, typically done by the author.
   - Recording updated status of defects (in formal reviews).
6. Follow-up:
   - Checking that defects have been addressed.
   - Gathering metrics.
   - Checking on exit criteria (for more formal review types).

Roles and Responsibilities

- Manager:
  - Decides on the execution of reviews, allocates time in project schedules and determines if the review objectives have been met.
- Moderator:
  - The person who leads the review of the document or set of documents, including planning the review, running the meeting, and follow-up after the meeting.

Roles and Responsibilities

- Author:
  - The writer or person with chief responsibility for the document (s) to be reviewed.
- Reviewers:
  - individuals with a specific technical or business background (also called checkers or inspectors) who, after the necessary preparation, identify and describe findings (e.g. defects) in the product under review.

Roles and Responsibilities

- Scribe (or recorder):
  - Documents all the issues, problems and open points that were identified during the meeting.

Types of Reviews

- A single software product may be the subject of more than one review. If more than one type of review is used, the order may vary. For example, an informal review may be carried out before a technical review, or an inspection may be carried out on a requirements specification before a walkthrough with customers.

Walkthrough

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

Technical Review

- Documented, defined defect-detection process that includes peers and technical experts with optional management participation
- May be performed as a peer review without management participation;
- Ideally led by trained moderator (not the author);
- Pre-meeting preparation by reviewers;
- Optional use of checklists,

Technical Review

- Preparation of a review report which includes list of findings, the verdict whether the software product meets its requirements and, where appropriate, recommendations related to the findings
- May vary in practice from quite informal to very formal;
- **Main purposes:** discuss, make decisions, evaluate alternatives, find defects, solve technical problems and check conformance to specifications and standards.

Inspection

- Led by trained moderator (not the author);
- Usually conducted as peer examination;
- Defined roles, Includes metrics;
- Formal process based on rules and checklists
- Specified enter and exit criteria for acceptance of the software product.
- Inspection report including list of findings;
- Formal follow-up process (with optional process improvement components)
- **Main purpose:** find defects.

Static Analysis by Tools

- The objective of static analysis is to find defects in software source code and software models. Static analysis is performed without actually executing the software being examined by the tool; dynamic testing does execute the software code.

Static Analysis by Tools

- The value of static analysis is:
  - Early detection of defects prior to test execution.
  - Early warning about suspicious aspects of the code or design, by the calculation of metrics, such as a high complexity measure.
  - Identification of defects not easily found by dynamic testing.
  - Detecting dependencies and inconsistencies in software models, such as links.
  - Improved maintainability of code and design.
  - Prevention of defects, if lessons are learned in development

Static Analysis by Tools

- Typical defects discovered by static analysis tools include:
  - Referencing a variable with an undefined value;
  - Inconsistent interface between modules and components;
  - Variables that are never used;
  - Unreachable (dead) code;
  - Missing and erroneous logic (potentially infinite loops)
  - Programming standards violations;
  - Security vulnerabilities;
  - Syntax violations of code and software models.

Test Design Techniques

The Test Development Process

- Expected results should be produced as part of the specification of a test case and include outputs, changes to data and states, and any other consequences of the test. If expected results have not been defined then a plausible, but erroneous, result may be interpreted as the correct one. Expected results should ideally be defined prior to test execution.

The Test Development Process

- During test implementation the test cases are developed, implemented, prioritized, and organized in the test procedure specification.
- The test procedure specifies the sequence of actions for the execution of a test.

Categories of Test Design Techniques

- This syllabus refers to specification-based test design techniques as black-box techniques and structure-based test design techniques as white-box techniques.
- Common features of specification-based techniques:
  - Models, either formal or informal, are used for the specification of the problem to be solved, the software or its components.
  - From these models test cases can be derived systematically.

Equivalence Partitioning

- Inputs to the software or system are divided into groups that are expected to exhibit similar behavior, so they are likely to be processed in the same way for both valid data and invalid data.
- Partitions can also be identified for outputs, internal values, time-related values and for interface parameters.
- Tests can be designed to cover all valid and invalid partitions.

Equivalence Partitioning

- Equivalence partitioning is applicable at all levels of testing.
- Equivalence partitioning can be used to achieve input and output coverage goals.
- It can be applied to human input, input via interfaces to a system, or interface parameters in integration testing.

Equivalence Class Partitioning

- Identify and group the factors that are to be tested and treated equivalently.
- Choose at least 1 option per partition that represents all values in same partition.
- For a simple range 23 to 58, 3 tests be executed.

Invalid | Valid | Invalid

←---22 | 23 ---- 58 | 59 --→

  - 1 value less than 23, e.g. 15
  - 1 value from 23 to 58, e.g. 45
  - 1 value above 58, e.g. 65

Boundary Value Analysis

- Behavior at the edge of each equivalence partition is more likely to be incorrect than behavior within the partition, so boundaries are an area where testing is likely to yield defects.
- The maximum and minimum values of a partition are its boundary values. A boundary value for a valid partition is a valid boundary value; the boundary of an invalid partition is an invalid boundary value.

Boundary Value Analysis

- Tests can be designed to cover both valid and invalid boundary values.
- Boundary value analysis can be applied at all test levels. It is relatively easy to apply and its defect finding capability is high.
- This technique is often considered as extension of equivalence partitioning or other black box test design techniques.

Boundary Value Analysis

- Example: Range 23 to 58
  - Boundary Values of Valid Partition – 23 and 58.
  - Values just below – 22 and 57.
  - Values just above – 24 and 59.
  - Series – 22, 23, 24, 57, 58, 59.

Invalid | Valid | Invalid

←-----**22** | **23** ------ **58** | **59** -----→

22, 23, 24, 57, 58, 59

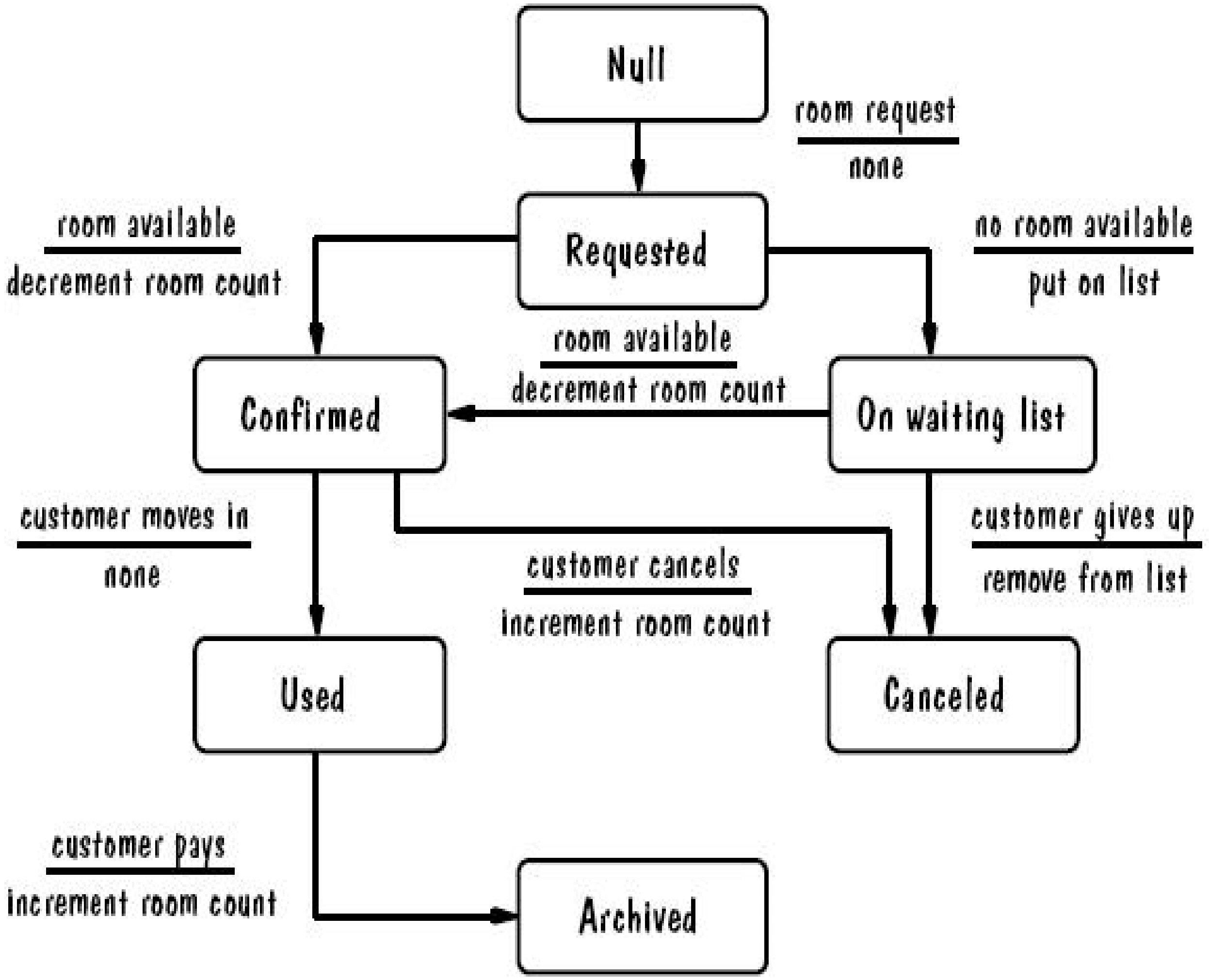
Decision Table Example

- Company X sells merchandise to wholesale and retail outlets. Wholesale customers receive a two percent discount on all orders.
- The company also encourages both wholesale and retail customers to pay cash on delivery by offering a two percent discount for this method of payment.
- Another two percent discount is given on orders of 50 or more units. Each column represents a certain type of order.

Decision Table Example

Order less than 50 Units		Y	Y	Y	Y	N	N	N	N
Cash on Delivery		Y	Y	N	N	Y	Y	N	N
Wholesaler?		Y	N	Y	N	Y	N	Y	N
Discount Rate	0%				X				
	2%		X	X					X
	4%	X					X	X	
	6%					X			

State Transition Testing



Use Case Testing

- Tests can be derived from use cases. A use case describes interactions between actors (users or system), which produce a result of value to a system user or customer.
- Each use case has preconditions, which need to be met for a use case to work successfully. Each use case terminates with post-conditions, which are the observable results and final state of the system after the use case has been completed. A use case usually has a mainstream (i.e. most likely) scenario, and sometimes alternative branches.

Use Case Testing

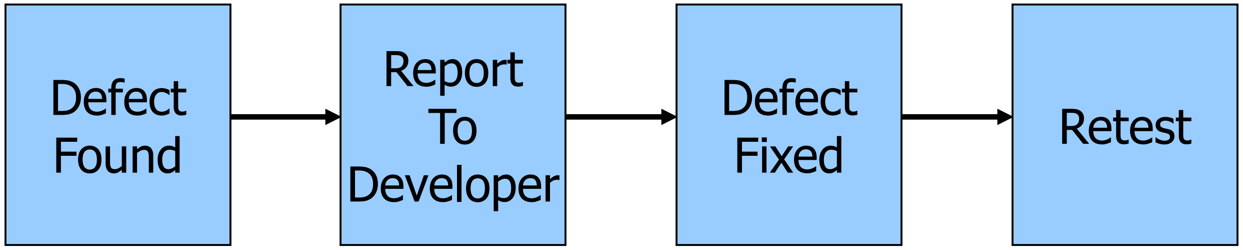
- Use cases describe the “process flows” through a system based on its actual likely use, so the test cases derived from use cases are most useful in uncovering defects in the process flows during real-world use of the system. Use cases, often referred to as scenarios, are very useful for designing acceptance tests with customer/user participation.

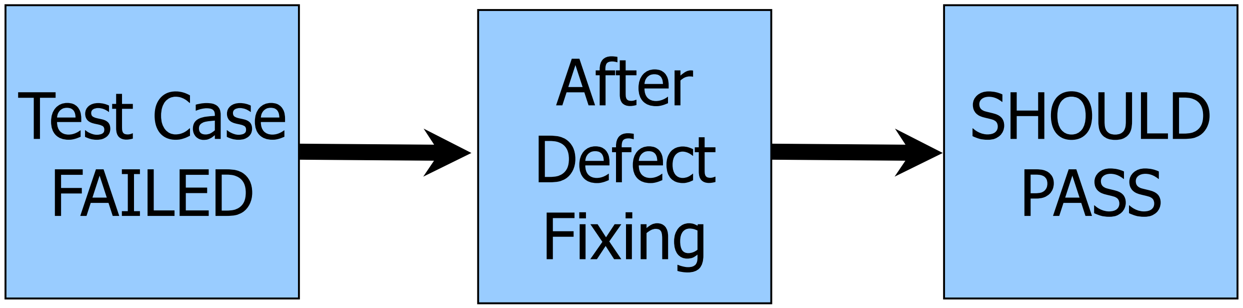
Types of Testing

Retesting

- Retesting is re-running the failed test case.
- Retesting is done after defect is found and fixed.
- It is done by re-executing the same (failed) test case after the fix is done.

Retesting

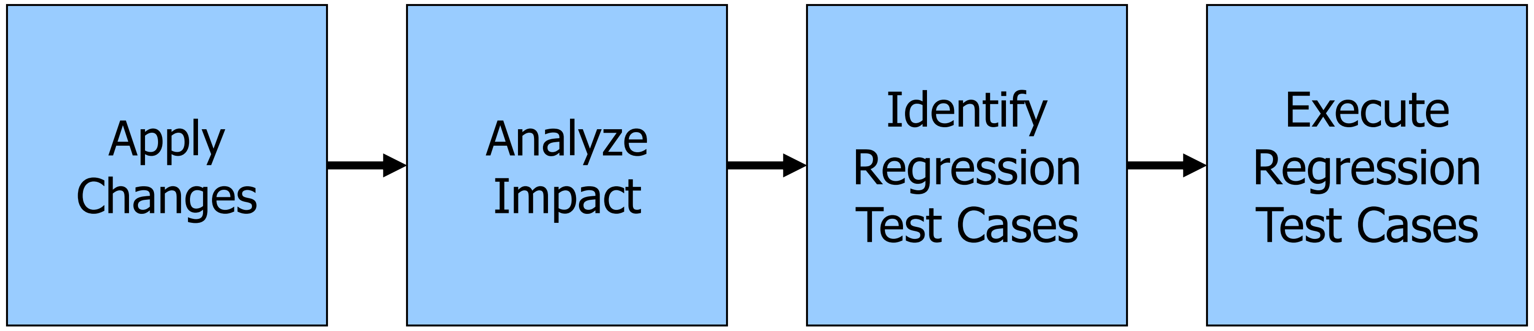

Defect Found
Report To Developer
Defect Fixed
Retest

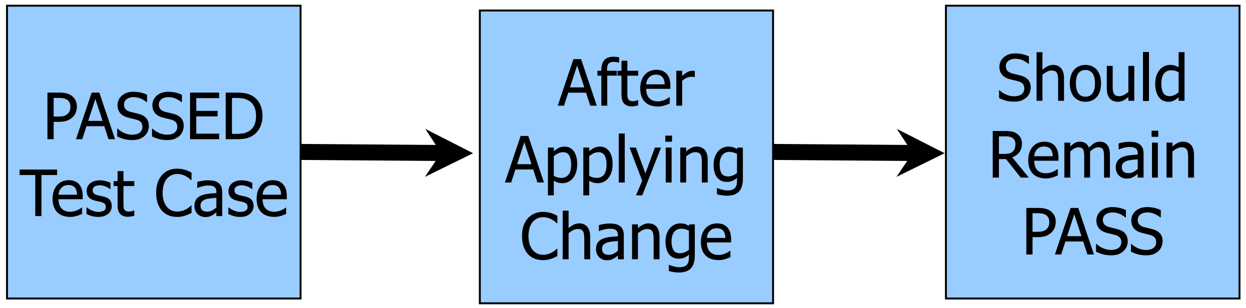

Test Case FAILED
After Defect Fixing
SHOULD PASS

Regression Testing

- Regression testing is the testing of the software after a modification has been made to ensure the reliability of each software release.
- Objective – to check changes did not introduce any new errors into the system.
- Regression testing is an ideal candidate for test automation because of its repetitive nature.

Regression Testing





Performance Testing

- Performance is also tested at the client / browser and server levels.
- The performance test is classified into below sub-tests.
  - Load Testing.
  - Stress Testing.

Load Testing

- Check the server behavior with peak load.
- Load Simulation.
- Load testing simulates the expected usage of a software program, by simulating multiple users that access the program's services concurrently.

Stress Testing

- Stress testing is testing that investigates the behavior of software (and hardware) under extraordinary operating conditions.
- Stress testing involves subjecting the program to heavy loads or stresses.
- Identify degradation point and crash point of the system.

Installation Testing

- To test installation procedure is documented and the people from installation team are trained.
- Installation testing is required to ensure:
  - Application is getting installed properly.
  - New program that is installed is working as desired.
  - Old programs are not hampered.
  - System stability is maintained.
  - System integrity is not compromised.

Configuration Testing

- Testing the software with the different possible hardware configurations.
- Objectives:
  - To determine Minimum and Optimal configuration of hardware that provides the required performance characteristics.
- Determine effect of adding / modifying hardware resources.
- Partially validate the application.

Compatibility Testing

- Compatibility testing is testing how well software performs in a particular hardware, software, operating system, or network environment.
- It is similar to multiplatform testing.
- More significant in case of web-based applications where any browser can be used to access the application.

Usability Testing

- Usability testing is testing for ‘ease of use’
- Clearly this is subjective and depends on the targeted end-user or customer.
- User interviews, surveys, video recording of user sessions and other techniques can be used.
- Programmers and developers are usually not appropriate as usability testers.

User Interface Testing

- This type of testing is performed to check how user-friendly the application is.
- To determine 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 assistance of system personnel

User Interface Testing

- Example:
  - Spelling check.
  - Graphic check.
  - Meaningful error messages.
  - Meaningful help documents (Manual support testing).
  - Accuracy of data displayed.