COURSE NAME

SOFTWARE QUALITY AND TESTING

CSC 4133

(UNDERGRADUATE)

CHAPTER 6

SOFTWARE TESTING

TESTING

- Testing is one of the most important parts and commonly performed activities in QA
- Basic idea of testing involves the execution of software and the observation of its behavior or outcome
- If a failure is observed, the execution record is analyzed to locate and fix the fault(s) that caused the failure. Otherwise, we gain some confidence that the software under test is more likely to fulfill its designated functions

TESTING THE SYSTEM

- Software testing is treated as a distinct process because it involves a variety of unique activities, techniques, strategies, and policies
- Testing is performed to reveal defects and show to what extent the software possesses different quality attributes, such as reliability and performance
- Testing begins almost at the same time a project is conceptualized
- Testing is carried out by different people at different stages of system development
- A number of different strategies/techniques can be applied at each level of testing
- A number of metrics can be monitored to measure the progress of testing
- Testing is influenced by organizational policies (developer vs individual tester)
- Testing can be performed as a combination of manual and automated modes of execution of test cases

PRINCIPLES OF TESTING

- Principle 1 – Testing shows presence of defects but cannot prove absence of defects
- Principle 2 – Exhaustive testing is impossible
  - Testing everything (all combinations of inputs and preconditions) is not feasible. Instead of exhaustive testing, risk analysis, time & cost analysis, and priorities should be used to focus testing efforts
- Principle 3 – Early testing
  - When defects are found earlier in the lifecycle, they are much easier and cheaper to fix
- Principle 4 – Defect clustering
  - A small number of modules usually contains most of the defects discovered during pre-release testing or is responsible for most of the operational failures.
  - There is NO equal distribution of defects within one test object. The place where defect occurs, it's likely to find some more. The testing process must be flexible and respond to this behaviour.

PRINCIPLES OF TESTING

Principle 5 – Pesticide paradox

- If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this "pesticide paradox", test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to find potentially more defects.

Principle 6 – Testing is context dependent

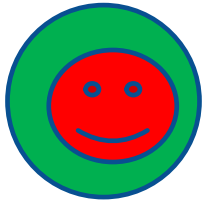
- Testing is done differently in different contexts, e.g. safety-critical software is tested differently from an e-commerce site.

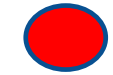
Principle 7 – Absence-of-errors fallacy

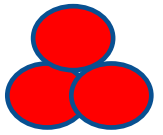
- Finding and fixing defects does not help if the system built is unusable and does not fulfil the users' needs and expectations.
- Just because testing didn't find any defects in the software, it does not mean that the software is ready to be shipped.

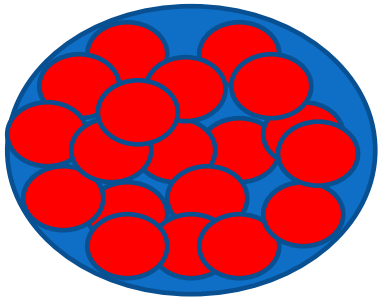
TESTING LEVELS

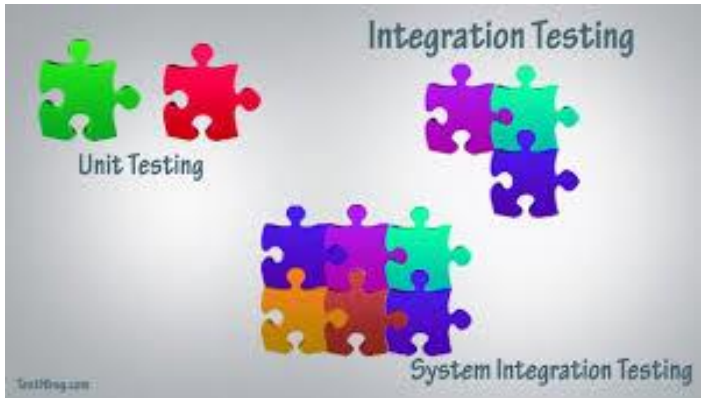
- Unit testing (Done by Developer)
  - Test Individual program units, such as procedure, methods in isolation
- Integration testing (Done by Tester)
  - Modules are assembled to construct larger subsystem and tested
- System testing (Done by Tester)
  - Test whole system which Includes wide spectrum of testing such as functionality, load
- Acceptance testing (Done by End-User)
  - Customer's expectations from the system











TESTING LEVELS

Regression Testing

- New test cases are not designed
- Tests are selected, prioritized and executed
- To ensure that nothing is broken in the new version of the software to accommodate any change

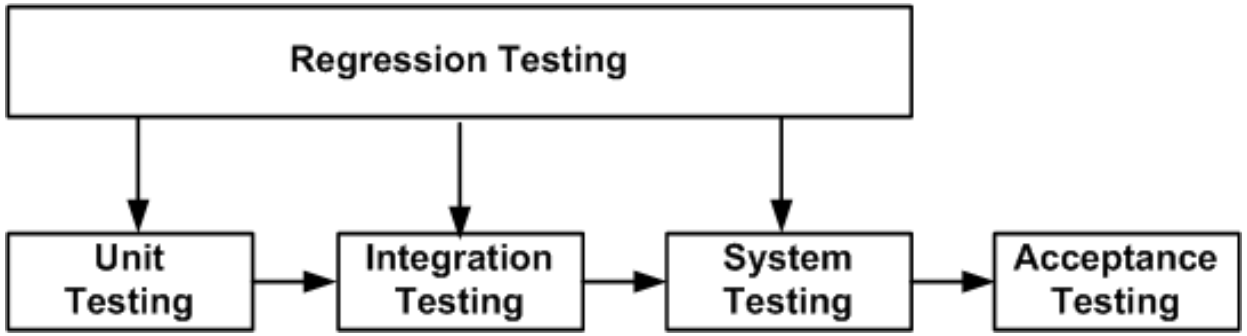
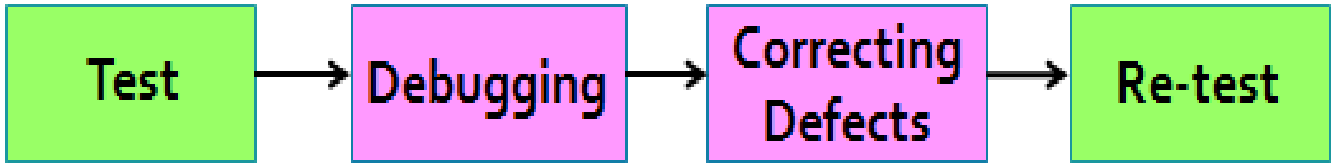


Fig: Regression testing at different software testing levels

TESTING & DEBUGGING



- Test and re-test are test activities
- Debugging and correcting defects are developer activities

Developer Roles & Responsibilities

- Implements requirements and develop structures
- Design and programs the software
- Creating a product is his success
- Perceptions are constructive
- Software is known and driven by delivery

Tester Roles & Responsibilities

- Plans testing activities
- Design test cases
- Concerned only on finding defects
- Finding errors made by developer is his success
- Perceptions are destructive
- Software is unknown (user) and driven by quality

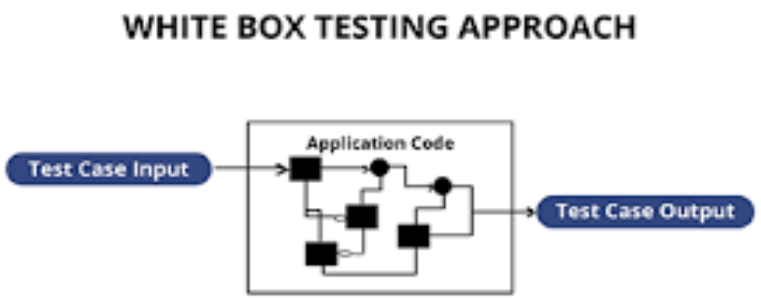
TESTING KEY QUESTION

WHY	▪ Demonstration of proper behavior or quality ▪ Defect-free software development (defect detection and removal)
HOW	▪ Techniques/activities/process/etc. ▪ Generic testing process - Test planning and preparation, Test Execution, Analysis and follow-up
VIEW	▪ Functional/external/black-box ▪ Structural/internal/white-box ▪ Gray-box (mixed black-box & white-box) testing
EXIT	▪ Functional Coverage (white-box) vs. Usage-based: quality/reliability goals

TESTING TECHNIQUES

White-box testing is Implementation-based also known as structural testing

- Examines source code with focus on:
  - (1) Control flow: refers to flow of control from one instruction to another
  - (2) Data flow: refers to propagation of values from one variable or constant to another variable
- Software developers perform structural testing on the individual program units they write
- Using white-box testing methods, you can derive test cases that:
  - (1) guarantee that all independent paths within a module have been exercised at least once
  - (2) exercise all logical decisions on their true and false sides
  - (3) execute all loops at their boundaries and within their operational bounds
  - (4) exercise internal data structures to ensure their validity



TESTING TECHNIQUES

Black-box testing is Specification-based also known as functional testing

- Examines the program that is accessible from outside at the external interface level of a system
- Applies the input to a program and observe the externally visible outcome
- It is applied to both an entire program as well as to individual program units
- Software quality assurance group(preferred) attempts to find errors in the following categories:

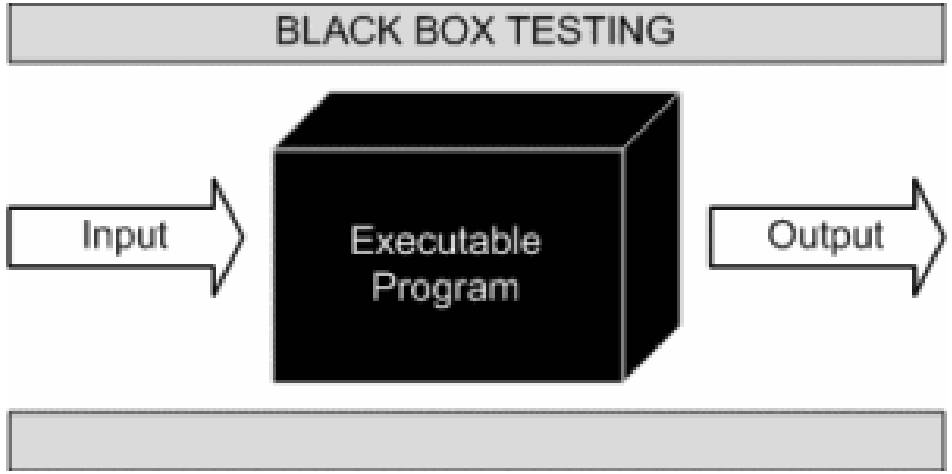
(1) incorrect or missing functions

(2) interface errors

(3) errors in external database access (accessibility)

(4) behavior or performance errors

(5) initialization and termination errors



TESTING TECHNIQUES

- High-level: Whole system ==> black-box (late in testing, e.g. system testing)
- Low –level: Individual statements, data, and other elements ==> white-box (test in small/early)
- Middle-levels of abstraction ==> Gray-box
  - Functional/subroutine/procedure, module , subsystem etc.
  - Method, class, super-class
- Gray-box (mixed black-box & white-box) testing:
  - Many of the middle levels of testing
  - Example: procedures in modules
    - Procedures individually as black box
    - Procedure interconnection ➔ white box at module level

WBT VS. BBT

- Perspective:
  - BBT views the objects of testing as a black-box while focusing on testing the input-output relations or external functional behavior
  - WBT views the objects as a glass-box where internal implementation details are visible & tested
- Objects:
  - WBT is generally used to test small objects (e.g., small software products or small units of large software products)
  - BBT is generally more suitable for large software systems or substantial parts of them as a whole
- Timeline:
  - WBT is used more in early sub-phases(e.g., unit and component testing)
  - BBT is used more in the late sub-phases (e.g., system and acceptance testing)

WBT VS. BBT

- Defect Focus and Fixing:
  - In BBT, failures related to specific external functions can be observed, leading to corresponding faults being detected & removed. *Emphasis* ==> Reduce chances of encountering functional problems by target customers.
  - In WBT, failures related to internal implementations can be observed, leading to corresponding faults being detected & removed directly. *Emphasis* ==> Reduce internal faults so that there is less chance for failures later on.
- Defect detection & Fixing:
  - Defects detected through WBT are easier to fix than those through BBT
  - WBT may miss certain types of defects (e.g., omission & design problems) which could be detected by BBT
  - *In general*: BBT is effective in detecting & fixing problems of interfaces & interactions, *while* WBT is effective for problems localized within a small unit.

WBT VS. BBT

- Techniques:
  - A specific technique is BBT if external functions are modeled
  - A specific technique is WBT if internal implementations are modeled

- Tester:
  - BBT is typically performed by dedicated professional testers, and could also be performed by third-party personnel in a setting of IV&V
  - WBT is often performed by developers themselves

WHEN TO STOP TESTING?

- Exit Criteria
  - Not finding (anymore) defects is NOT an appropriate criteria to stop testing activities (Why?!? → User Acceptance is matter at the end of the project development)
- Resource-based criteria: A decision is made based on resource consumptions
  - Stop when you run out of time
  - Stop when you run out of money
  - Such criteria are irresponsible , as far as product quality is concerned
- Quality-based criteria:
  - Stop when quality goals reached
    - usability, reliability, resemble actual customer usages
  - Other substitute: Activity completion ("stop when you complete planned test activities")

MANUAL TESTING VS. AUTOMATED TESTING

Manual Testing

- Oldest and most rigorous type of software testing
- Requires a tester to perform manual test operations
  - Hard to repeat
  - Not always reliable
  - Costly
  - Time consuming
  - Labour intensive

Automated Testing

- Testing employing software tools
- Execute tests without manual intervention
  - Fast
  - Repeatable
  - Reliable
  - Reusable
  - Programmable
  - Saves time

TEST AUTOMATION

- Key issues related to Test Automation
  - Specific needs and potentials for automation and selection of existing testing tools, if available/suitable
  - Availability of user training for these tools and time/effort needed
  - Overall cost, including costs for tool acquisition, support, training, and usage
  - Impact on resource, schedule, and project management
- Pre-requisites for test automation
  - The system is stable and its functionalities are well defined
  - The test cases to be automated are unambiguous
  - The test tools and infrastructure are in place
  - The test engineers have prior successful experience with automation
  - Adequate budget should have been allocated for the procurement of tools

TEST AUTOMATION

- Which Tests/Test cases to automate?
  - Fully automated testing is not possible, test automation can NOT replace manual testing
  - Tests that should be run for every build of the application (repetitive execution, e.g. regression)
  - Data tests that use multiple data values for the same inputs (e.g. data-driven test)
  - Tests that require detailed information from the application internals (e.g., GUI attributes)
  - Tests to be used for stress or load testing

TEST AUTOMATION

- Automated testing CANNOT replace Manual testing
- Which Test/Test cases should NOT be automated?
  - Usability testing ("How easy is the application to use?") cannot be performed by automation
  - Logical error often cannot be identified by the automated testing.
  - Specification (e.g. SRS), Design document cannot be tested by automation.
  - One-time testing, "ASAP" testing –"we need to test NOW!"
  - Ad hoc/random testing: based on intuition, expertise, and knowledge of application
  - Tests without predictable results

REFERENCES

- Software Testing And Quality Assurance – Theory and Practice - Kshirasagar Naik & Priyadarshi Tripathy
- Software Quality Engineering: Testing, Quality Assurance and Quantifiable Improvement - Jeff Tian