# Software Testing

# Testing Concepts

# Background

- Main objectives of a project: High Quality & High Productivity (Q&P)
- Quality has many dimensions
  - reliability, maintainability, interoperability etc.
- Reliability is perhaps the most important
- Reliability: The chances of software failing
- More defects => more chances of failure => lesser reliability
- Hence Q goal: Have as few defects as possible in the delivered software

# Faults & Failure

- Failure: A software failure occurs if the behavior of the s/w is different from expected/specified.
- Fault: cause of software failure
- Fault = bug = defect
- Failure implies presence of defects
- A defect has the potential to cause failure.
- Definition of a defect is environment & project specific

# Role of Testing

- Reviews are human processes - can not catch all defects
- There will be requirement defects, design defects and coding defects in code
- These defects have to be identified by testing
- Therefore testing plays a critical role in ensuring quality.
- All defects remaining from before as well as new ones introduced have to be identified by testing.

# Detecting defects in Testing

- During testing,  software under test (SUT) executed with set of test cases
- Failure during testing => defects are present
- No failure => confidence grows, but can not say “defects are absent”
- To detect defects, must cause failures during testing

# Test Oracle

- To check if a failure has occurred when executed with a test case, we need to know the correct behavior
- I.e. need a test oracle, which is often a human
- Human oracle makes each test case expensive as someone has to check the correctness of its output

# Test case and test suite

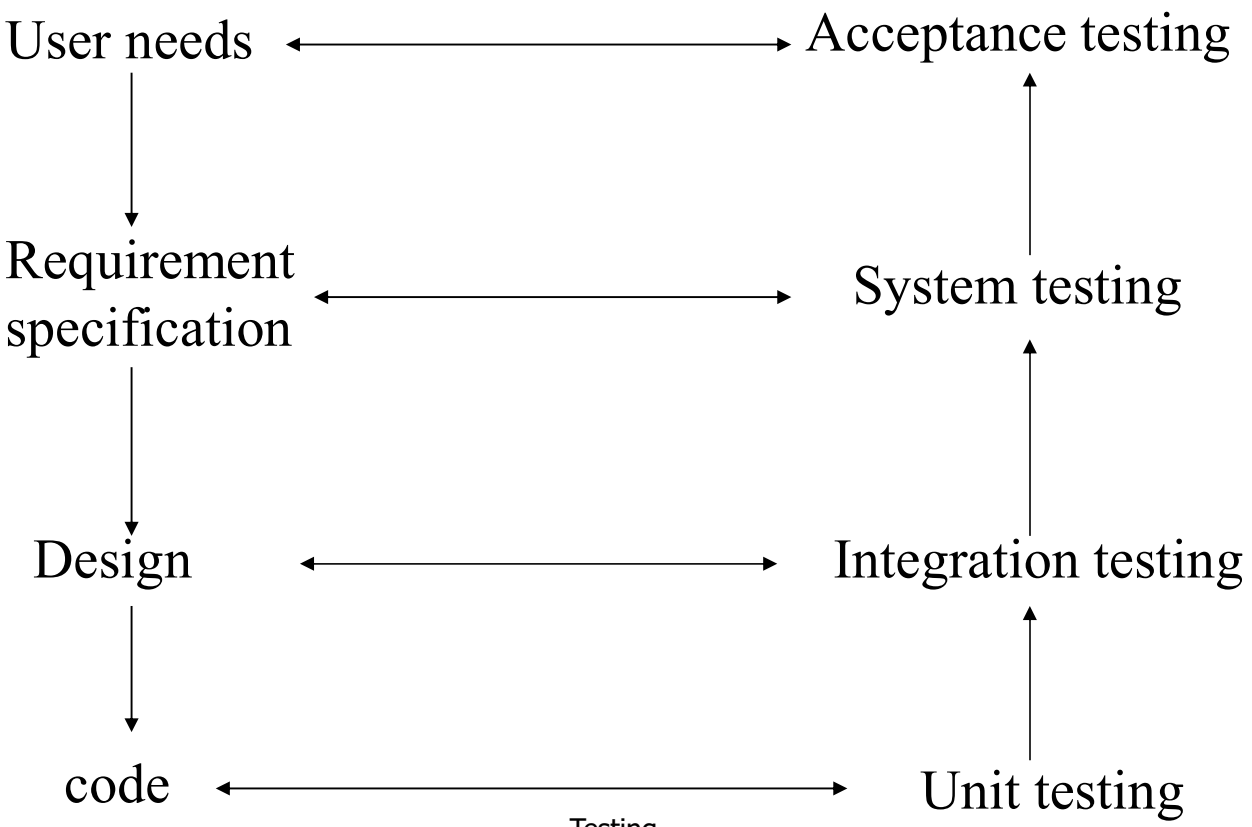
- Test case – a set of test inputs and execution conditions designed to exercise SUT in a particular manner
  - Test case should also specify the expected output – oracle uses this to detect failure
- Test suite - group of related test cases generally executed together

# Test harness

- During testing, for each test case in a test suite, conditions have to be set, SUT called with inputs, output checked against expected to declare fail/pass
- Many test frameworks (or test harness) exist that automate the testing process
  - Each test case is often a function/method
  - A test case sets up the conditions, calls the SUT with the required inputs
  - Tests the results through assert statements
  - If any assert fails – declares failure

# Levels of Testing

- The code contains requirement defects, design defects, and coding defects
- Nature of defects is different for different injection stages
- One type of testing will be unable to detect the different types of defects
- Different levels of testing are used to uncover these defects


User needs
Acceptance testing
Requirement specification
System testing
Design
Integration testing
code
Unit testing

# Unit Testing

- Different modules tested separately
- Focus: defects injected during coding
- Essentially a code verification technique
- UT is closely associated with coding
- Frequently the programmer does UT;
- coding phase sometimes called “coding and unit testing”

# Integration Testing

- Focuses on interaction of modules in a subsystem
- Unit tested modules combined to form subsystems
- Test cases to “exercise” the interaction of modules in different ways
- May be skipped if the system is not too large

# Integration Testing

- Develop the integration plan by examining the structure chart :
  - big bang approach
  - top-down approach
  - bottom-up approach
  - mixed approach

# Big Bang Integration Testing

- Big bang approach is the simplest integration testing approach:
  - all the modules are simply put together and tested.
  - this technique is used only for very small systems.

# Big Bang Integration Testing

- Main problems with this approach:
  - If an error is found:
    - It is very difficult to localize the error
    - The error may potentially belong to any of the modules being integrated.
  - Debugging errors found during big bang integration testing are very expensive to fix.

# Bottom-up Integration Testing

- Integrate and test the bottom level modules first.
- A disadvantage of bottom-up testing:
  - when the system is made up of a large number of small subsystems.
  - This extreme case corresponds to the big bang approach.

# Top-down integration testing

- Top-down integration testing starts with the main routine:
  - and one or two subordinate routines in the system.
- After the top-level 'skeleton’ has been tested:
  - immediate subordinate modules of the 'skeleton’ are combined with it and tested.

# Mixed Integration Testing

- Mixed (or sandwiched) integration testing:
  - uses both top-down and bottom-up testing approaches.
  - Most common approach

# Integration Testing

- In top-down approach:
  - testing waits till all top-level modules are coded and unit tested.
- In bottom-up approach:
  - testing can start only after bottom level modules are ready.

# System Testing

- Entire software system is tested
- Focus: does the software implement the requirements?
- Validation exercise for the system with respect to the requirements
- Generally the final testing stage before the software is delivered
- May be done by independent people
- Defects removed by developers
- Most time consuming test phase

# Acceptance Testing

- Focus: Does the software satisfy user needs?
- Generally done by end users/customer in customer environment, with real data
- Only after successful AT software is deployed
- Any defects found,are removed by developers
- Acceptance test plan is based on the acceptance test criteria in the SRS

# Other forms of testing

- Performance testing
  - tools needed to “measure” performance
- Stress testing
  - load the system to peak, load generation tools needed
- Regression testing
  - test that previous functionality works alright
  - important when changes are made
  - Previous test records are needed for comparisons
  - Prioritization of testcases needed when complete test suite cannot be executed for a change