Unit 2
Test Case Design Strategies

Test case Design Strategies - Using Black Box Approach to Test Case Design - Boundary Value Analysis - Equivalence Class Partitioning - State based testing - Cause-effect graphing - Compatibility testing - user documentation testing - domain testing - Random Testing - Requirements based testing - Using White Box Approach to Test design - Test Adequacy Criteria - static testing vs. structural testing - code functional testing - Coverage and Control Flow Graphs - Covering Code Logic - Paths - code - complexity testing - Additional White box testing approaches-Evaluating Test Adequacy Criteria.

SMART TESTER

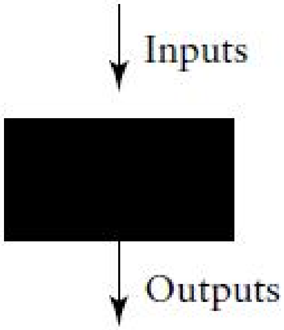
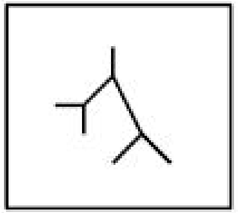
- Design tests that
  - reveal defects, and
  - can be used to evaluate software performance, usability, and reliability.
- Plan for testing,
- select the test cases, and
- monitor the process to insure that the resources and time allocated for the job are utilized effectively.

- Novice testers, taking their responsibilities seriously, might try to test a module or component using all possible inputs and exercise all possible software structures.
- The goal of the smart tester is to understand the functionality, input/output domain, and the environment of use for the code being tested. For certain types of testing, the tester must also understand in detail how the code is constructed

- Knowledge of type of defect injected
- Intelligently select the test inputs - greatest probability of detecting defects
- chose carefully to maximize use of resources

TEST CASE DESIGN STRATEGIES

- Effective test case
  - a greater probability of detecting defects,
  - a more efficient use of organizational resources,
  - a higher probability for test reuse
  - closer adherence to testing and project schedules and budgets
  - the possibility for delivery of a higher-quality software product

Test Strategy	Tester's View	Knowledge Sources	Methods
Black box		Requirements document Specifications Domain knowledge Defect analysis data	Equivalence class partitioning Boundary value analysis State transition testing Cause and effect graphing Error guessing
White box		High-level design Detailed design Control flow graphs Cyclomatic complexity	Statement testing Branch testing Path testing Data flow testing Mutation testing Loop testing

- The smart tester knows that to achieve the goal of providing users with low-defect, high-quality software, *both* of these strategies should be used to design test cases.

RANDOM TESTING

- Randomly selects inputs from the domain
- Valid input domain – 1 to 100 randomly pick 55, 24, 3

Issues

- Are the three values adequate to show that the module meets its specification when the tests are run? Should additional or fewer values be used to make the most effective use of resources?
- Are there any input values, other than those selected, more likely to reveal defects? For example, should positive integers at the beginning or end of the domain be specifically selected as inputs?

- Should any values outside the valid domain be used as test inputs? For example, should test data include floating point values, negative values, or integer values greater than 100?

Thank you