Lec 5 Validation and Verification

Objectives

- [ ] Black-box testing techniques
    - [ ] Equivalence Class Testing
    - [ ] Boundary Value Analysis
    - [ ] Requirements Based Testing
- [ ] White-box testing techniques
    - [ ] Statement Testing
    - [ ] Branch Testing

IEEE Standard Test Definitions:
Test case → a set of inputs, execution conditions, and a pass/fail criterion

Test case specification → a requirement to be satisfied by one or more actual test cases

Test obligation → a partial test case specification, requiring some property deemed important to thorough testing

Test suite → a set of test cases

Test or test execution → activity of executing test cases and evaluating their results

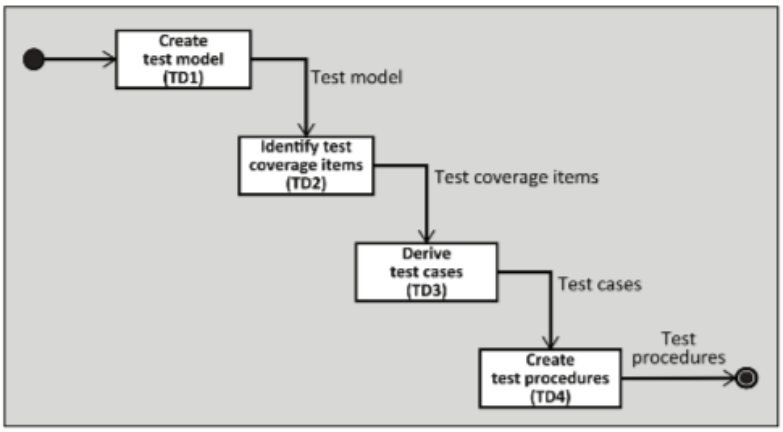
Adequacy criterion → is a predicate that is true (satisfied) or false (not satisfied) of a <program, test suite> pair.

Obligation and criteria give you guidelines to define

- Which testing techniques to use?
- What stopping criterion?
- A checklist of testing activities
- Driver by test requirements
    - Aiming at measuring the quality of testing (Verification of verification)
- Givens you a method with guidelines and techniques to be:
    - Adapted on a project-by-project basis;
    - Improved using a feedback loop (e.g. capability levels)

Test design and implementation process

- **Test item** → work product that is being tested (ISO/IEC/IEEE 29119-1)
- **Test Model** → represents testable aspect of a test item
    - Such as function, transaction, feature, quality attribute, or structural element identified as a basis for testing
    - Reflects required test completion criterion in the test strategy
- **Test Coverage item**s → attributes of the test model that can be covered during testing.
- **Test case** → a set of preconditions inputs (including actions, where applicable), and expected results, developed to determine whether or not the covered part of the test item has been implemented correctly.



Coverage = (N/T*100)%
C = Coverage achieved by a specific test design technique
N = Number of test coverage items covered by executed test cases
T = Total number of test coverage items identified by the test design technique

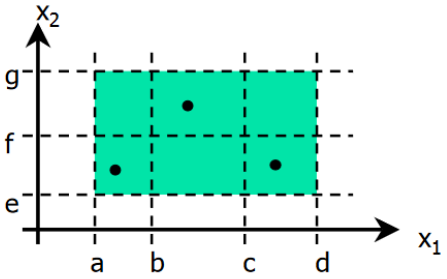
---

Specification-based techniques
Equivalence class partitioning (black box technique):

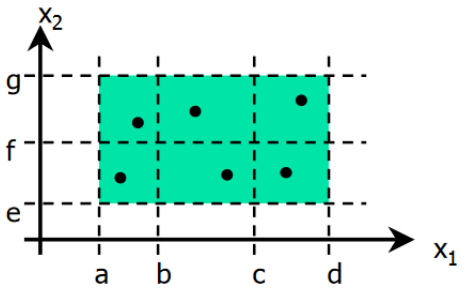
- Input data is divided into partitions of valid and invalid values → requires that all partition exhibit the same behaviour
- Test coverage items → partitions
- E..g an application accepts an int >100
    - Valid equivalence class: int >100
    - Invalid equivalence classes: int <=100, decimals, alpha non-numeric

Types of Equivalence partitioning testing:

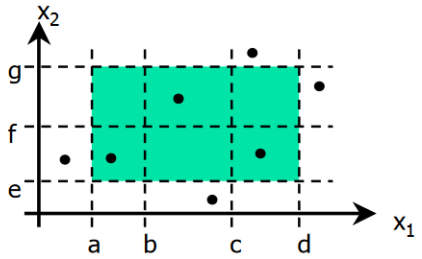
- **Weak Normal Equivalence Class testing**
    - One variable from each valid equivalence class is tested
    - **No. test cases =**
      max / [[v : 1 .. #variables
      • number_equivalence_classes (variable v)]



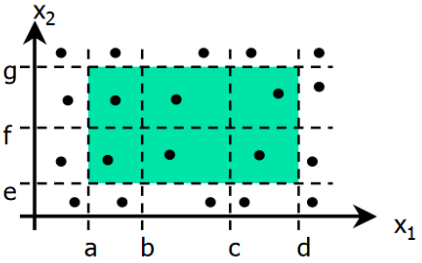
- **Strong Normal Equivalence Class testing**
    - Test cases from each element of the Cartesian product of equivalence classes
    - **Completeness:** covers all equivalence classes test each possible combinations of inputs
    - **No. test cases =**
      X / [[v : 1 .. #variables
      • number_equivalence_classes (variablev)]]



- **Weak Robust Equivalence Class Testing**
  - Weak normal equivalence but test invalid inputs (e.g. string instead of integer)
  - **No. test cases =**
    max / [[v : 1 .. #variables
    • number_equivalence_classes (variablev)]]
    +
    +/ [[v : 1 .. #variables
    • number_invalid_bounds (variablev)]]



- **Strong Robust Equivalence Class Testing**
  - Test all valid and invalid elements of the product of the equivalence classes
  - **No. test cases =**
    × / [[v : 1 .. #variables
    • number_equivalence_classes (variablev)
    + number_invalid_bounds (variablev)]



https://www.softwaretestinggenius.com/equivalence-class-testing-black-box-software-testing-techniques/

ECT advantages

- Reduce the number of test cases → reduces test execution time
- Helps identify significant/meaningful test cases, systematically
- Can be applied to all levels of testing (unit testing, integration testing, system testing, etc)

ECT disadvantages

- Does not consider the conditions for boundary value
- Identification of equivalence classes relies heavily on the expertise of testers
- The coverage metric is only significant w.r.t the identification of equivalence classes

Exercises (1/3)

Define equivalence classes for the following function:

- Int16_t calculator (int8_t a, int8_t b, operator_t op)
    - Operator_t is an enum with values: plus, minus, multiply, divide
    - Calculator simply returns the arithmetic operation: a op b
    - If the operation cannot be performed correctly, returns NaN

Determine test cases for a 100% coverage with:

- A Weak Normal Equivalence Class Testing
- A Strong Robust Equivalence Class Testing

Solutions

Inputs equivalence classes:

- **int8_t a [-128 to 127]**
- **int8_t b [-128 to 127]**
- **Operator_t op:** plus, minus, multiply, divide

A Weak Normal Equivalence Class Testing:

- Requires one variable from each equivalence class to be tested
- As Operator_t op contains 4 equivalence classes; there will be 4 test cases

a	**b**	**op**	**results**
5	-1	plus	4
5	0	minus	5
5	3	multiple	15
2	4	divide	0

A Strong Robust Equivalence Class Testing:

- Test all valid and invalid elements of the product of the equivalence classes
- Total number of test cases are 2*4*5

Exercises (2/3)

Consider the following function:

- def triangleType (a,b,c)
- Determines the type of triangle (isocele, equilateral, scalene) given the length of its three sides a,b and c. The range of the length of the sides lies between 10 and 50 (both inclusive)

- How many test cases would you need for
    - A Weak Normal Equivalence Class Testing
    - A Strong Normal Equivalence Class Testing
    - A Weak Robust Equivalence Class Testing
    - A Strong Robust Equivalence Class Testing

Solutions

7 invalid classes and 1 valid classes

Invalid class:

1. a<10
2. a>50
3. b<10

4. b>50
5. c<10
6. c>50
7. a>=b+c
8. b>=a+c
9. c>=a+b

Valid class:

1. 10<=a,b,c<=50

A Weak Normal Equivalence Class Testing

- One variable from each valid equivalence class is tested, as there is only 1 valid class, this means there is 1 test

A Strong Normal Equivalence Class Testing

- Test cases from each element of the Cartesian product of equivalence classes
    - 1 for strong normal equivalence class testing , as there is only 1 valid class, so don't need to find the cartesian product

A Weak Robust Equivalence Class Testing

- Weak normal equivalence but test invalid inputs
- Invalid classes = 1,2,3,4,5,6 and the valid classes
    - 7 Total classes

A Strong Robust Equivalence Class Testing

- Test all valid and invalid elements of the product of the equivalence classes
- Invalid classes: a has set {1,2}, b has set {2,3}, c has set {5,6}. Each variable has 2 sets each. The cartesian product is 2 * 2 * 2 = 8
- For each of these situations, you need to consider 7,8,9. So the test cases will be 8 * 3 = 24
- Invalid classes + valid class = 24 + 1 = 25 test cases

Boundary Value Analysis (black box technique):

- Test along the boundaries
- Based on the **single fault assumption**: Failures are rarely the product of two or more simultaneous faults

Types of Boundary Value Analysis

- **Two-value boundary testing**
    - One value on the boundary
    - One value an incremental distance outside the boundary of the equivalence partition
- **Three-value boundary testing**
    - One value on the boundary
    - One value an incremental distance on one side the boundary of the equivalence partition
    - One value an incremental distance on the other side the boundary of the equivalence partition
- **Non-standard Two-value boundary testing**
    - One value on the boundary
    - One value an incremental distance inside the boundary of the equivalence partition

Exercise

Consider the following function:

- def triangleType(a,b,c)
- determine the type of triangle (isosceles, equilateral, scalene) given the length of its three sides a, b, and c. The range of the length of the sides lies between 10 and 50 (both inclusive)

How many test cases would you need for a two-value boundary analysis?

Solution

Two-value boundary testing

- One value on the boundary values (10, 50) for a,b,c
- One value an incremental distance outside the boundary of the equivalence partition values (9,51) for a,b,c
- So 4 values for 3 parameters = 4* 3 = 12 test cases
- Could include 1 additional test case for nominal a=30, b=30, c=30

Requirements-based testing (black box technique):

- Assumes requirements are decomposed into “atomic” requirements
    - Atomicity is not defined
        - Leaves in a requirements decomposition process, making sure these leaves cannot be decomposed into testable sub-requirements
        - Testable requirements (scenario, stories, BDD features)
- Test cases are defined to check if an atomic requirement is met
- **Advantages**:
    - Helps with the requirements traceability (verification cross-reference matrix)
    - Focuses on tests with added value for the business
- **Disadvantages**
    - Assumes an excellent definition of requirements (clear, testable, atomic, non-redundant, complete, correct, …)
    - Focus on expected behaviour (not invalid cases)

Structure-based techniques

Statement Testing (White-box technique):

- Test model : source code
- Test coverage items: statements in the source code
- It covers all the paths, lines, and *statements* of a source code

Advantages

- Simple to understand/implement
- Makes sure every single line of code is executed by (at least) a test

Disadvantages

- May lead to implement useless test cases (e.g. getters and setters)
- Does not consider conditions coverage (defined latter)

Branch Testing (White-box testing)

- Tests each branch where decisions are made
- **Branch Coverage**
    - **Objective**: minimum number of paths which will ensure all paths are covered.
    - Measures the percentage of decisions outcomes that have been tested

Advantages

- Simple to understand/implement
- Makes sure every single line of code is executed (at least) a test

DIsadvantages

- May lead to implement useless test cases (e.g. getters and setters)
- Does not consider condition covered