Software Testing

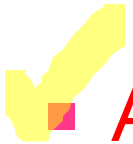
Testing Techniques & OO Testing

Software Testing: Techniques

Software Testing

- A process of executing a program with the specific intent of finding an error prior to delivery to the end user.
- A successful test is one that uncovers an as-yet-undiscovered error.

A Good Test



- A good test has a high probability of finding an error.
- A good test is not redundant.
- A good test should be neither too simple nor too complex.

Software Testing

- Formal reviews/inspections by themselves cannot locate all software defects (faults).
- Testing is an essential part of the quality assurance work and a normal part of modern software engineering.

Exhaustive and Selective Testing

- Exhaustive testing is not possible for most real applications.
  - Too many logic paths and two many input data combinations!
- Test parts of a system which are commonly used rather than those which are rarely executed.

Test Data and Test Cases

- Test data
  - Inputs which have been devised to test the system.
- Test cases
  - Inputs to test the system and the predicted outputs from these inputs if the system operates according to its specification

Test Case Design

Test Case Design

- Generation of test cases is based on the level of testing to be performed and the particular testing techniques.
- The number of test cases processed is less important than the quality of the test cases used in software testing.

Test Case Design Approaches

- White-box test cases design
- Black-box test cases design

White-Box Testing Approach

White-Box Testing

- Focuses on the program control structure.
  - **Structural Testing**
- Code-based techniques
- Derivation of test cases according to program structure.
- Knowledge of the program is used to identify test cases.

White-Box Testing

- Objective is to exercise all program statements and logical conditions at least once.
  - Not all path combinations!
- White-box testing is important, since there are many program defects that black-box testing can not uncover.

White-Box Testing Techniques

Basis Path Testing

- A white-box testing
- Code-based technique
- Path testing
- Analyze number of paths in program & Decide which ones to test.
- The objective is to ensure that the set of test cases is such that each path through the program is executed at least once.

Constructing Program Flow Graph

- Convert into a program flow graph that shows nodes representing program decisions and arcs representing the flow of control.

Program Flow Graph

- Flow graph notations:
  - Nodes
  - Edges
  - Regions
  - Predicate nodes

Deriving Basis Path Set

- Derive the set of linearly independent tests that ensure the coverage.
- Although all paths are executed, all combinations of paths are not executed.

Basis Path Set

- An execution path
- An independent path
- A basis path set

Cyclomatic Complexity

- A software metric that provides a quantitative measure of the logical complexity of a program.
- McCabe's observation:
  - The program complexity is directly related to the complexity of control flow or the number of branches in the program.

Cyclomatic Complexity

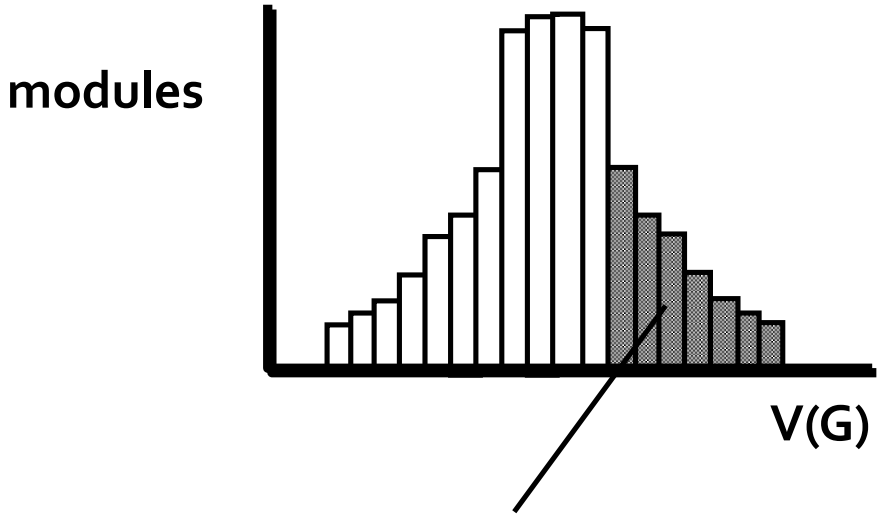
- Cyclomatic complexity is defined as
  - The number of edges – The number of nodes +2
  - The number of regions of the flow graph
  - The number of closed regions of the flow graph + 1
  - The number of predicates + 1

Cyclomatic Complexity

- The cyclomatic complexity number is the number of independent paths in the basis path set of a program.
- An upper bound for the number of tests that must be conducted!
  - To ensure that all statements have been executed at least once!

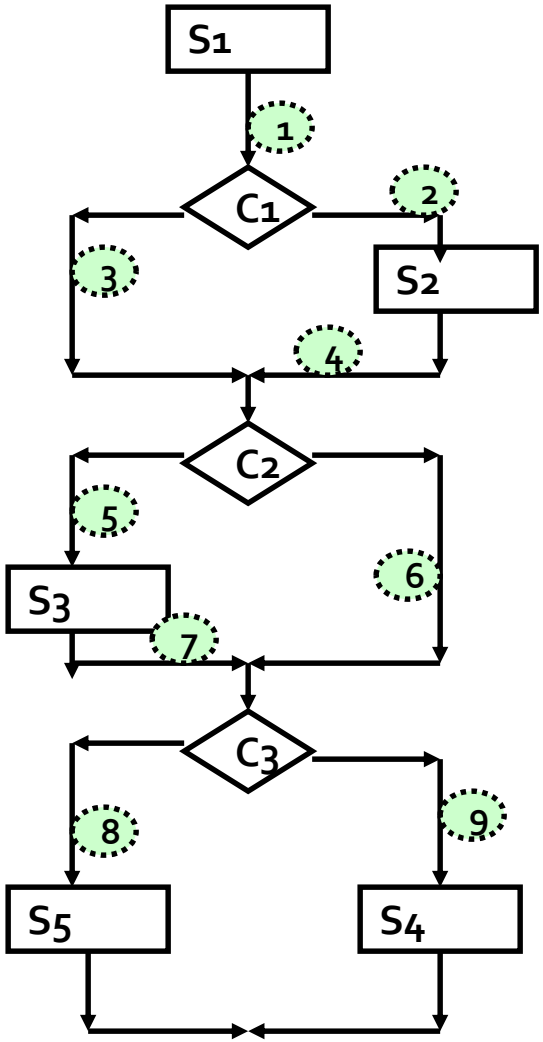
Cyclomatic Complexity

A number of industry studies have indicated that the higher V(G), the higher the probability of errors!



Modules in this range are more error prone!

Example: Cyclomatic Complexity

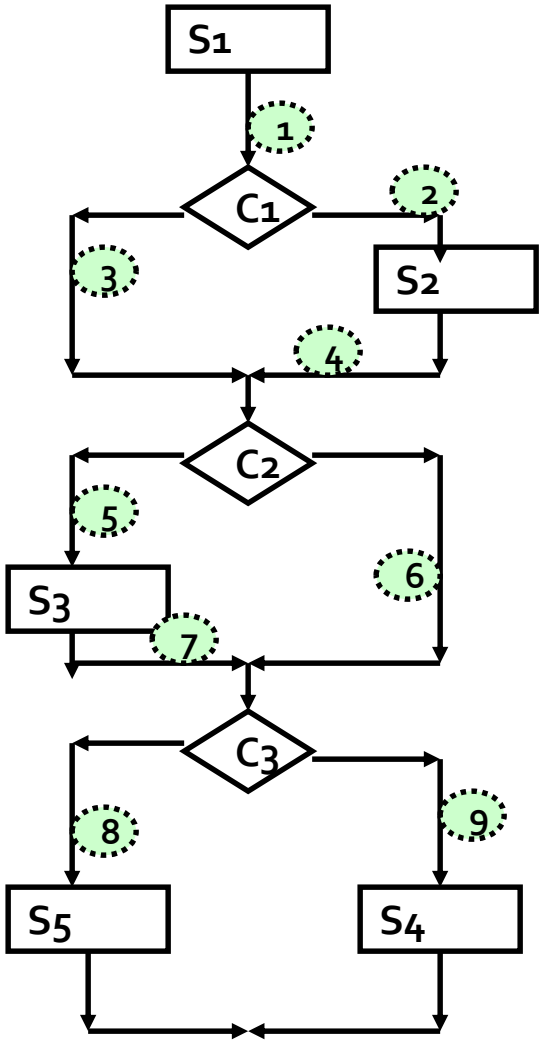


Since for each binary decision, there are 2 paths and there are 3 in sequence, there are 2^3 = 8 total "logical" paths

path1 : S1-C1-S2-C2-C3-S4
path2 : S1-C1-S2-C2-C3-S5
path3 : S1-C1-S2-C2-S3-C3-S4
path4 : S1-C1-S2-C2-S3-C3-S5

path5 : S1-C1-C2-C3-S4
path6 : S1-C1-C2-C3-S5
path7 : S1-C1-C2-S3-C3-S4
path8 : S1-C1-C2-S3-C3-S5

Example: Cyclomatic Complexity



Since for each binary decision, there are 2 paths and there are 3 in sequence, there are 2^3 = 8 total "logical" paths

path1 : S1-C1-S2-C2-C3-S4
path2 : S1-C1-S2-C2-C3-S5
path3 : S1-C1-S2-C2-S3-C3-S4
path4 : S1-C1-S2-C2-S3-C3-S5

path5 : S1-C1-C2-C3-S4
path6 : S1-C1-C2-C3-S5
path7 : S1-C1-C2-S3-C3-S4
path8 : S1-C1-C2-S3-C3-S5

How many Linearly Independent paths are there?
Cyclomatic number = 3 decisions +1 = 4

***One set* would be:**
path1 : includes segments (1,2,4,6,9)
path2 : includes segments (1,2,4,6,8)
path3 : includes segments (1,2,4,5,7,9)
path5 : includes segments (1,3,6,9)

Deriving Test Cases

1. Construct a flow graph.
2. Determine the cyclomatic complexity.
3. Determine a basis set of linearly independent paths.
4. Prepare test cases that will force execution of each path in the basis set.

Control Structure Testing

- Basis path testing is one form of control structure testing.
- Code-based techniques
- Three others for further exercise of the program logic:
  - Condition testing
  - Data flow testing
  - Loop testing

Condition Testing

- A test case design method that exercises the logical conditions contained in a program.

Dataflow Testing

- A test case design method that selects test paths of a program according to the locations of definitions and uses of variables in the program.

Loop Testing

- A test case design method that focuses on the validity of loop constructs of varying degrees of complexity.

Black-Box Testing Approach

Black-Box Testing

- An approach to testing where the program is considered as a 'black-box'.
- **Functional Testing**
- To validate functional requirements without regard to the internal workings of a program.
- Test planning can begin early in the software process.

Black-Box Testing

- Specification-based techniques
- Devise a set of data inputs that fully exercise all functional requirements for a program based on the specification.
- The test designer has no knowledge of algorithm implementation.
- The test cases are designed from the requirement statements directly.

Black-Box Testing Techniques

- Equivalence class partitioning
- Boundary-Value analysis

Equivalence Class Partitioning

- The input domain is subdivided into a collection of subsets, or equivalent classes, which are deemed equivalent according to a specified relation, and a representative set of tests (sometimes only one) is taken from each class.

Equivalence Class Partitioning

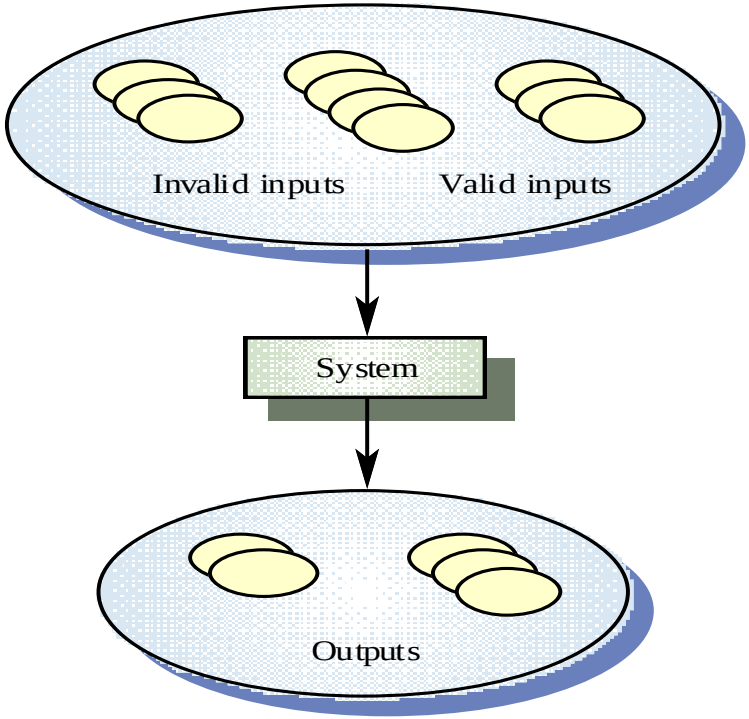
- Specification-based technique
- Input data and output results often fall into different classes where all members of a class are related.
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member.

Equivalence Partitioning

- A black-box testing method.
- Divides the input domain of a program into classes of data that are likely to exercise specific software function.
- Equivalence partitions are sets of test cases where the program should behave in an equivalent way.
- Test cases should be chosen from each partition.

Sample Equivalence Classes

- Valid input data
- Invalid input data



Example: Equivalence Partitioning

- If input is a 5-digit integer between 10,000 and 99,999,
  - equivalence partitions are <10,000, 10,000-99,999 and > 99,999.
- Choose test cases from three equivalence classes.

Boundary Value Analysis

- Specification-based technique
- Test cases are chosen on and near the boundaries of the input domain of variables, with the underlying rationale that many faults tend to concentrate near the extreme values of inputs.

Boundary Value Analysis

- A black-box test
- A greater number of errors tend to occur at the boundaries of the input domain rather than in the center!
  - Past experiences show that "Boundaries" are error-prone!

Boundary Value Analysis

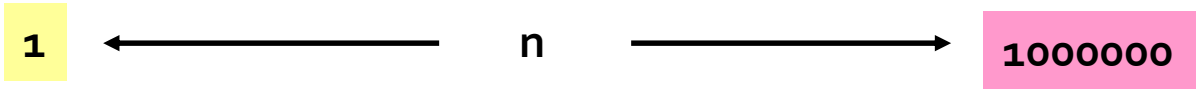
- Probes the program's ability to handle data at boundaries.
- Do equivalence-class partitioning, add test cases for boundaries (*at boundary*, *outside*, *inside*).
- BVA leads to a selection of test cases that exercise boundary values.

Example: BVA

- If input is a 5-digit integer between 10,000 and 99,999,
  - equivalence partitions are <10,000, 10,000-99,999 and > 99,999.
- Choose test cases at the boundary of these sets.
  - 00000, 09999, 10000, 99999, 10001

Example: BVA

1 <= number of employees, **n** <= 1000000



1 <= employee **age** <= 150

The "basic" boundary value testing for a value would include:

1. - at the "minimum" boundary
2. - immediately above minimum
3. - between minimum and maximum (nominal)
4. - immediately below maximum
5. - at the "maximum" boundary

White-Box vs Black-Box Testing

- Black-box testing is complementary to white-box testing.
  - White box testing as testing in the small!
  - Black-box testing as testing in the large!
- Both black-box testing and white-box testing are necessary to test a program thoroughly.

Testing for Specialized Environments

- Several specialized testing situations:
  - GUI's,
  - Client/server architectures
  - Real-time systems

Object-Oriented Testing

Object-Oriented Testing

- Testing strategy changes
  - The concept of the ‘unit’ broadens due to encapsulation.
  - Integration focuses on classes and their execution across a ‘thread’ or in the context of a usage scenario.
  - Validation uses conventional black box methods.

Unit Testing

- No longer effective to test a single operation in isolation!
- The lowest testable unit should be the encapsulated class or object.

Class Testing

- Class testing is the equivalent of unit testing.
- The relevant operations are exercised and the state of the class is examined.

Class Testing

- Unit test methods:
  - **Random** test method
  - **Partition** test method

Random Testing Method

- A variety of different operation sequence is generated randomly (but valid).

Partition Testing Method

- Input and output are categorized and test cases are designed to exercise each category.
- Reduces the number of test cases required to test a class in much the same way as equivalence partitioning for conventional software!

State-Based Partitioning

- Categorizes class operations based on their ability to change the state of the class.

Attribute-Based Partitioning

- Categorizes class operations based on attributes that they use.

Category-Based Partitioning

- Categorizes class operations based on the generic function that each performs.

Inter-Class Testing

- Integration test methods:
  - **Random** testing method
  - **Partition** testing method
  - **Behavior** testing method
    - **Based on the state transition**

Validation Testing

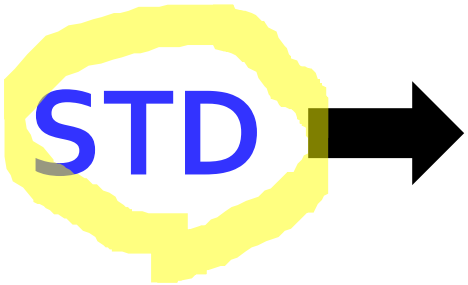
- Black box testing
- Scenario-based testing driven by the use cases.

Software Testing Documentation

Software Testing Documentation

- Test Plan
- Test Design Specification
- Test Case Specification
- Test Procedure Specification
- Test Log
- Test Summary Report
- Traceability Matrix

Software Test Document (STD)



- ✓Test Plan
- ✓Test Case Design
  - Acceptance Test
- ✓Traceability Matrix