Black Box Testing Categories

- Equivalence class partitioning
- Boundary value analysis

Equivalence Class Partitioning

- Done to eliminate redundant test cases
- Divide inputs into set of classes
  - All inputs in a class are “equivalent for testing purposes
  - Test just one input from each class
- Most useful when requirements are expressed as set of conditions
- Example: Age groups
  - Between 0 - 120

Table 10.1 Equivalence Class Example

Class	Representative
Low	-5
0–12	6
13–19	15
20–35	30
36–120	60
High	160

Boundary Value Analysis

- Extension of EQP, deals only with data that can be sorted and organized in intervals
- Motivation is that problems often occur at boundaries of range
  - Have test cases of boundary – 1, boundary, boundary + 1
  - Can often reduce number of cases by assuming boundary falls between numbers

Class	All Cases	In Class	Reduced
Low	-1, 0	-1	-1
0 - 12	-1, 0, 1, 11, 12, 13	0, 1, 11, 12	0, 12
13 - 19	12, 13, 14, 18, 19, 20	13, 14, 18, 19	13, 19
20 - 35	19, 20, 21, 34, 35, 36	20, 21, 34, 35	20, 35
36 - 120	35, 36, 37, 119, 120, 121	36, 37, 119, 120	36, 120
High	121	121	121

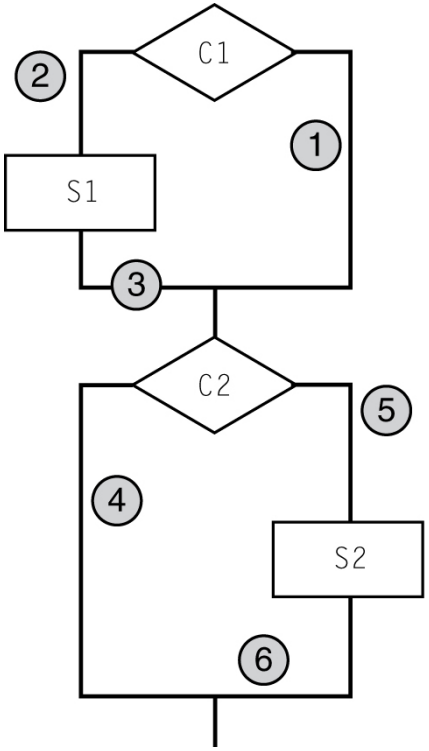
White Box Testing

- Want to ensure good code coverage
  - More coverage means more test cases
  - Perception of quality increases as test cases increase
- Path analysis
  - Determine how many paths are in a program
    - Limit loops to one iteration
  - Decide which ones to test

White Box Testing (cont.)

- Coverage hierarchy (strongest to weakest)
  - All **logical paths**
  - All **linearly independent paths** (or **basis paths**)
    - Set from which all paths may be constructed by a linear combination
    - This equals the McCabe Cyclomatic Complexity number (number of binary decisions + 1)
  - **Branch** coverage
    - Test each branch out of every binary decision, ensures each edge in flow graph is taken
  - **Statement** coverage (minimum level)
    - Test every statement

White Box Testing (cont.)

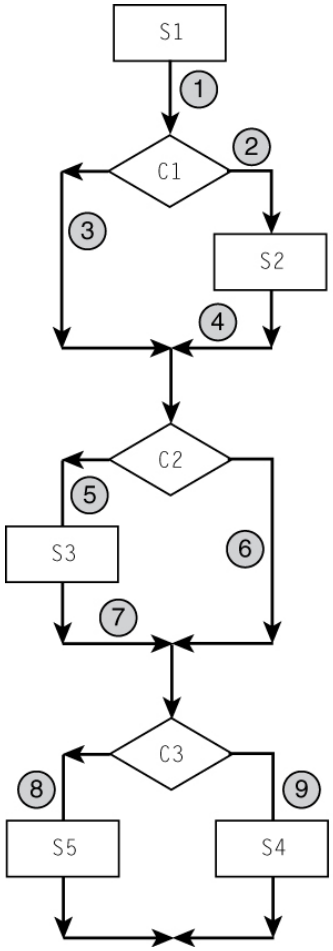


Consider Path1, Path2, and Path3 as the Linearly Independent Set

	1	2	3	4	5	6
Path1	1				1	1
Path2	1			1		
Path3		1	1	1		
Path4		1	1		1	1

- All paths = 4
- Linearly independent paths = 3
- Branch coverage = 2
- Statement coverage = 1

White Box Testing (cont.)



Because 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?
Using 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)
```

- All paths =8
- Linearly independent paths = 4
- Branch coverage = 2
- Statement coverage = 2

Determining LI Paths

- Recall McCabe's Cyclomatic Complexity number
  - E = number of edges in graph
  - N = number of nodes in graph
  - Cyclomatic number = E – N + 2
  - CN = number of closed regions + 1
  - = number of branching nodes + 1
  - = number of **linear independent paths**

Determining LI Paths (cont.)

- To find a set of LI paths
  - **Path 1:** set all conditions to true
  - **Path 2:** set first condition to false
  - **Path 3:** set first condition back to true, set second condition to false

    .

    .

    .

  - **Path last:** last condition is false, all others are true

Path Testing Example

```
int computeFines (int daysLate[], int numDVD, Boolean printOn)
{
int MAX_FINE_PERIOD = 21, MAX_FINE = 42, DAILY_FINE = 2;
int cumFine = 0; int i;
for i = 0; i < numDVD; i++) {
      fine = MAX_FINE;
      if (daysLate[i] <= MAX_FINE_PERIOD) {
            fine = daysLate[i] * DAILY_FINE;
      }
      logFine (fine);
      if (printOn == TRUE) {
            printFine (fine);
      }
      cumFine += fine;
}
return cumFine;
}
```

Sample Method Testing Checklist

- Verify operation at normal parameter values **B**
- Verify operation at limit parameter values **B**
- Verify operation outside parameter values **B**
- Ensure all instructions execute **S**
- Check all paths (both sides of branches) **P**
- Check use of all called methods **W**
- Verify the handling of all data structures **W**
- Verify the handling of all files **W**
- Check normal operation of all loops **W**

Sample Method Testing Checklist

- Check abnormal operation of all loops **W**
- Check normal termination of all recursions **W**
- Check abnormal operations of all recursions **W**
- Verify handling of all error conditions **G**
- Check timing and synchronization **G**
- Verify all hardware dependencies **G**

Inspections and Reviews

- **Review**: process involving humans
  - Reading and understanding a document
  - Analyzing the document to find errors
- **Walkthrough**: author explaining a document to others
- **Software inspection**: detailed reviews of document following set methodology
  - Team of 3 – 6
  - Author included
  - Other members working on related aspects (designer, coder, tester, support) to motivate

Software Inspection Process

- Planning
  - Select team and moderator, make sure document is ready for review
- Overview
- Preparation in advance
- Examination meeting
  - Reader summarizes parts of work
  - Focused only on finding errors
- Rework to correct errors
- Follow-up
  - Moderator approves changes
  - Or re-inspect

Testing vs. Inspections

- Testing
  - Cheaper to find errors but more expensive to correct
  - Only works with code
  - Catches errors late in process
  - Objective
  - Needed to estimate quality
- Inspections
  - Expensive since involves more people
  - Works with all kinds of documents
  - Catches errors earlier (if applied earlier than code development)
  - Subjective
  - Educates team

Static Analysis

- Examine static structure of documents
- Automated process using tools
  - Traceability
  - Coding guidelines
  - Security practices
- Applies to
  - Design documents
  - Source code
  - Executables
- Usually tools generate lots of false positives
  - Output needs to be checked and verified

SE Code of Ethics – Management

- Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
  - Ensure good management for any project on which they work, including effective procedures for promotion of quality and reduction of risk.
  - Ensure that software engineers are informed of standards before being held to them.
  - Ensure that software engineers know the employer's policies and procedures for protecting passwords, files and information that is confidential to the employer or confidential to others.

SE Code of Ethics – Management (cont.)

- Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
  - Assign work only after taking into account appropriate contributions of education and experience tempered with a desire to further that education and experience.
  - Ensure realistic quantitative estimates of cost, scheduling, personnel, quality and outcomes on any project on which they work or propose to work, and provide an uncertainty assessment of these estimates.
  - Attract potential software engineers only by full and accurate description of the conditions of employment.
  - Offer fair and just remuneration.

SE Code of Ethics – Management (cont.)

- Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
  - Not unjustly prevent someone from taking a position for which that person is suitably qualified.
  - Ensure that there is a fair agreement concerning ownership of any software, processes, research, writing, or other intellectual property to which a software engineer has contributed.
  - Provide for due process in hearing charges of violation of an employer's policy or of this Code.

SE Code of Ethics – Management (cont.)

- Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
  - Not ask a software engineer to do anything inconsistent with this Code.
  - Not punish anyone for expressing ethical concerns about a project.

Patriot Missile System