Software Validation and Testing

Part 1
Confusion

- The terminology is not standardized.
- Some people use “verification” to mean one of:
  - validation
  - testing
Verification vs. Validation: Our Definitions

- **Verification**: using formal techniques and tools to assess correctness of the product with respect to specifications.

- **Validation**: establishing that the software developed correctly capture the user’s needs and intent.
Validation vs. Verification vs. Testing

- Client perceived needs
  - Requirements specification
    - Formal specification of software
      - Actual software
        - Client Satisfaction

- Validation
  - Formal specification
  - Verification
  - Testing
Sometimes heard...

- **Verification**: Are we building the product right?

- **Validation**: Are we building the right product?

- Apparently due to Barry Boehm (the spiral model guy).
Validation may involve Testing

- **Testing**: discovering errors in the product, for purposes of eradicating them. Also, enhancing confidence in the product when no errors found.

- **Verification**: using specifications, logical techniques and tools to assess correctness of the product.

- The two can be mutually supportive, in ways to be discussed.
Testing is Necessary, but not really sufficient
Recall Famous Dijkstra Quote:

“Testing can show the presence of errors, but never their absence.”

- Rare exception: When the set of all inputs is finite and of a reasonable size, exhaustive testing can be used.

- Exhaustive testing is only feasible for small systems, typically hardware units, that are finite-state machines.
Exhaustive Testing

- Suppose we wanted to test a 32-bit multiply routine exhaustively. How long would it take?

- \(2^{32} \times 2^{32} = 2^{64}\) input combinations at, say, 1 combination per nanosecond

- about 585 years
Verification alone is not sufficient either

- Creating a formal specification is hard, sometimes even harder than developing the software.

- A formal specification seldom captures all needs and intent.

- It is difficult to ascertain that all needs are covered until the system is built.

- Therefore, validation may not be complete without testing in addition to verification.
Problems Attributable to Lack of Testing

- **Costly (dollars and lives) catastrophic failures:**
  - Therac 25
  - Ariane 5
  - Mars Polar Orbiter
  - and many more ($59.5 Billion in 2003 -- NIST)

- **Worms, viruses, and other infestations** (adware, spyware, who-knows-what-ware)
Standard Testing Terminology

- **Unit testing** tests self-contained units: classes, methods, functions, procedures.

- **Integration testing** tests combinations of units, such as packages, that have already been unit-tested, by having them mutually form an environment similar to actual use.

- **System testing** is top-level integration testing.

- **Acceptance testing** tests the final product according to pre-agreed criteria of the customer.
Regression Testing

- **Regression testing**: when changes are made, re-test previous test cases to ascertain that no new errors were introduced in the changes.

- "Smoke test": A coarse form of regression test to determine that the product doesn’t simply crash as a result of recent changes. [Apply to see if there is any “smoke” (sign of new errors).]
Mutation Testing

- A scheme for testing tests, by gauging their effectiveness of a given test.

- Assume that we have a test T which the code passes.

- The code is subjected to “mutation”. If the mutated code also passes test T, then T is less-likely to be regarded as a good test.
"V": model: A possible incorporation of testing in the software life-cycle

"Validate" requirements

"Verify" design

"Verify" program units

OPERATION & MAINTENANCE

ACCEPTANCE TESTING

SYSTEM TESTING

UNIT & INTEGRATION TESTING

PROGRAM DESIGN

SYSTEM DESIGN

REQUIREMENTS ANALYSIS

CODING
“Fault” vs. “Failure” Terminology

- A fault is an error in the code.
- A failure is the manifestation of a fault at runtime.
- The mapping from failures to faults is many-to-one.
- (A flaw is an error in the design.)
Typical Fault Profile
(faults/KLOC)

vs.

Development phase

Thousands of lines of source code in the entire project.
Relative **Cost** of Fixing Errors vs. Phase (typical)

(Ambler)

No scale given

(Boehm)
An Example

- The “Y2K Problem” probably would have cost 1/1000 (ignoring inflation) as much to fix at design/coding time as it actually took to fix in the field.

- (Then there is the cost of “fixing the fixes”.)
Testing Approaches

- Test your own code
- Test code of other people in your group
- One person dedicated to testing
- Outside testing group
- Independent testing company
- Cleanroom approach
Programmers do not test, or otherwise execute, their code.

Instead, programmers establish correctness by formal reasoning methods and construction techniques.

The actual testing takes place in the integration phase, which is done by a different team from the programmers.
Approach testing as an intellectual challenge in its own right

- Think as if testing someone else’s program, not your own.

- The objective is to find as many distinct problems as possible.

- Remember that you are testing the program and not the person.
Things Testing Can’t/Shouldn’t Do

- Don’t expect testing in itself to improve a poor design.

- As a developer, don’t rely on testing by others as the prime method for identifying your own mistakes.
Testing Example (1)

- A reputable programmer has produced a binary search method for searching an array:

```c
int search(float* a, int M, int N, float sought);
```

The method is supposed to determine the least index in the range of indices $M$ to $N-1$ where the value sought would be inserted, assuming that the array is to be maintained in increasing order. It is to be assumed that search will be called with $M \leq N$. If the element is greater than any in the array, the returned value should be $N$.

- Suggest how you would test this method, examining any trade-offs.
Another reputable programmer has produced class definition for a **circular buffer**: an implementation of a queue that uses a wrap-around array rather than a linked-list. (The purpose of wrap-around is so that elements of the array can be reused without shifting the array, which could take time proportional to the number of elements in the array.)

- The constructor of the circular buffer specifies an upper bound on the number of items to be stored.
- The primitives are boolean enqueue(Item), Item dequeue(), and boolean isEmpty().
- The return value of enqueue() is false if the buffer is already full.

Suggest how you would test this class, examining any trade-offs.
Testing Terminology

- **Black-box** (or opaque box) testing tests software against the operating environment, without using knowledge of internal structure.
  Also called: **Functional Testing**

- **White-box** (or clear-box) testing uses *knowledge of internal structure* to test specific pieces.
  Also called: **Structural Testing**
Testing Terminology

- **Gray-box**: Functional Testing assisted by structural knowledge to reduce unnecessary blind testing.
Testing Continuum

<table>
<thead>
<tr>
<th>Pure white-box (structural)</th>
<th>Gray-box (some structural)</th>
<th>Pure black-box (behavioral)</th>
<th>Live tests</th>
</tr>
</thead>
</table>

Development activity

Testing activity

Technical support, beta testing
Black- vs. White Box

- **Black-box** focuses on the specification: What is in the spec that the program doesn’t do?

- **White-box** focuses on the program: What does the program do that is not in the spec?

- If possible, don’t rely on one or the other exclusively.
Basic Testing Tools

- **Plan & Checklists**: tests to perform
  - With each test, pre-condition, post-condition
- **Test matrix/spreadsheet**: listing tests against use cases and potential error areas
- **Logging capability**
  - Note pad
  - Keystroke recorder (for playback)
  - Event logger
  - Screen recorder
  - Video camera
- **Tracking Database**
# Testing Matrix

<table>
<thead>
<tr>
<th>Tested by</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Test 4</th>
<th>Test 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case 1</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use Case 2</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use Case 3</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Test Result Categorization

- Area of defect
- Severity level of defect
- Follow up:
  - Responsibility
  - Time to fix
  - Lines of code affected
10 Sample Severity Levels, with examples (Boris Beizer)

- **Mild**: misspellings in output
- **Moderate**: misleading or redundant behavior
- **Annoying**: truncated names, etc.
- **Disturbing**: some transactions not processed
- **Serious**: lost transaction
- **Very serious**: incorrect output
- **Extreme**: frequent “very serious” errors
- **Intolerable**: data corrupted
- **Catastrophic**: shutdown
- **Infectious**: shutdown spreading to others
4 Possible Reaction Levels to failed tests

- **Defer**: fix as time permits
- **Schedule**: fix by some future date
- **Required**: fix before acceptance
- **Immediate**: fix before testing is continued
Testing Forms and Tracking

- Discrepancy report form
- Error investigation form
- Error reporting/tracking system/database
Discrepancy Report Form
(Sherry Pfleeger)

DISCREPANCY REPORT FORM

DRF Number:__________________________________________  Tester name:____________________________
Date: ___________________________________  Time: ______________________________
Test Number: ______________________________
Script step executed when failure occurred: __________________________________________________________
Description of failure: ________________________________________________________________________
_______________________________________________________________________________________
_______________________________________________________________________________________
_______________________________________________________________________________________
Activities before occurrence of failure:
_______________________________________________________________________________________
_______________________________________________________________________________________
Expected results:
_______________________________________________________________________________________
_______________________________________________________________________________________
Requirements affected:
_______________________________________________________________________________________
_______________________________________________________________________________________
Effect of failure on test:
_______________________________________________________________________________________
_______________________________________________________________________________________
Effect of failure on system:
_______________________________________________________________________________________
_______________________________________________________________________________________
Severity level: (LOW) 1 2 3 4 5 (HIGH)
## FAULT REPORT

**ORIGINATOR:** Joe Bloggs  
**BRIEF TITLE:** Exception 1 in dps_c.c line 620 raised by NAS  
**FULL DESCRIPTION:**  
Started NAS endurance and allowed it to run for a few minutes. Disabled the active NAS link (emulator switched to standby link), then re-enabled the disabled link and CDIS exceptioned as above. (I think the re-enabling is a red herring.)

### ASSIGNED FOR EVALUATION TO:

<table>
<thead>
<tr>
<th>CONFIGURATION ID</th>
<th>ASSIGNED TO</th>
<th>PART</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpo_s.c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CATEGORISATION:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Design</th>
<th>Spec</th>
<th>Docn</th>
</tr>
</thead>
</table>

### SEND COPIES FOR INFORMATION TO:

**EVALUATOR:**

**DATE:** 8/7/92

### CONFIGURATION ID

**ITEMS CHANGED**

<table>
<thead>
<tr>
<th>CONFIGURATION ID</th>
<th>IMPLEMENTOR/DATE</th>
<th>REVIEWER/DATE</th>
<th>BUILD/ISSUE NUM</th>
<th>INTEGRATOR/DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpo_s.c v.10</td>
<td>AWJ 8/7/92</td>
<td>MAR 8/7/92</td>
<td>6.120</td>
<td>RA 8-7-92</td>
</tr>
</tbody>
</table>

### COMMENTS:

dpo_s.c appears to try to use an invalid CID, instead of rejecting the message. AWJ

### CLOSED

**FAULT CONTROLLER:**

**DATE:** 9/7/92
## Testing Issue Spreadsheet

<table>
<thead>
<tr>
<th>Date Open</th>
<th>Sev</th>
<th>Risk</th>
<th>Owner</th>
<th>Estimated</th>
<th>Configuration</th>
<th>Close Date</th>
<th>Resolution</th>
<th>Root Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/21/99</td>
<td>1</td>
<td>1</td>
<td>Muhammad Zam</td>
<td>8/12/99</td>
<td>A.X.0.0001</td>
<td>8/3/99</td>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>7/21/99</td>
<td>2</td>
<td>4</td>
<td>Chuck Chavall</td>
<td>7/28/99</td>
<td>A.X.0.0001</td>
<td>7/30/99</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>7/24/99</td>
<td>3</td>
<td>9</td>
<td>Jenny Chung</td>
<td>8/12/99</td>
<td>A.Y.0.0001</td>
<td>8/1/99</td>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>7/23/99</td>
<td>3</td>
<td>9</td>
<td>Muhammad Zam</td>
<td>7/24/99</td>
<td>A.Z.0.0001</td>
<td>8/2/99</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>7/24/99</td>
<td>1</td>
<td>4</td>
<td>Chuck Chavall</td>
<td>7/31/99</td>
<td>B.Y.0.0001</td>
<td>8/1/99</td>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>7/24/99</td>
<td>2</td>
<td>5</td>
<td>Jenny Chung</td>
<td>8/17/99</td>
<td>C.X.0.0001</td>
<td>8/7/99</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>7/20/99</td>
<td>4</td>
<td>1</td>
<td>Frank Currant</td>
<td>8/14/99</td>
<td>C.Z.0.0001</td>
<td>8/5/99</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>7/22/99</td>
<td>5</td>
<td>1</td>
<td>Chuck Chavall</td>
<td>8/18/99</td>
<td>D.Z.0.0001</td>
<td>8/5/99</td>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>7/22/99</td>
<td>1</td>
<td>1</td>
<td>Jenny Chung</td>
<td>7/28/99</td>
<td>A.X.0.0001</td>
<td>8/4/99</td>
<td>Documentation</td>
<td>Functional</td>
</tr>
<tr>
<td>7/23/99</td>
<td>2</td>
<td>4</td>
<td>Muhammad Zam</td>
<td>8/17/99</td>
<td>A.X.0.0001</td>
<td>8/7/99</td>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>7/23/99</td>
<td>1</td>
<td>3</td>
<td>Chuck Chavall</td>
<td>7/26/99</td>
<td>A.X.0.0001</td>
<td>8/2/99</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>7/21/99</td>
<td>2</td>
<td>1</td>
<td>Jenny Chung</td>
<td>8/13/99</td>
<td>A.X.0.0001</td>
<td>8/5/99</td>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>7/23/99</td>
<td>1</td>
<td>2</td>
<td>Lary Kanemetsu</td>
<td>7/27/99</td>
<td>A.X.0.0001</td>
<td>8/7/99</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>7/24/99</td>
<td>2</td>
<td>3</td>
<td>Frank Currant</td>
<td>7/25/99</td>
<td>A.Y.0.0001</td>
<td>8/7/99</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>7/24/99</td>
<td>3</td>
<td>4</td>
<td>Chuck Chavall</td>
<td>8/13/99</td>
<td>A.Z.0.0001</td>
<td>8/2/99</td>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>7/24/99</td>
<td>4</td>
<td>1</td>
<td>Jenny Chung</td>
<td>8/19/99</td>
<td>C.Z.0.0001</td>
<td>8/1/99</td>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>7/19/99</td>
<td>1</td>
<td>2</td>
<td>Muhammad Zam</td>
<td>7/27/99</td>
<td>C.X.0.0001</td>
<td>7/30/99</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>7/22/99</td>
<td>2</td>
<td>4</td>
<td>Jenny Chung</td>
<td>8/18/99</td>
<td>C.Y.0.0001</td>
<td>8/1/99</td>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>7/21/99</td>
<td>3</td>
<td>5</td>
<td>Lary Kanemetsu</td>
<td>7/22/99</td>
<td>D.Z.0.0001</td>
<td>7/29/99</td>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>7/20/99</td>
<td>4</td>
<td>3</td>
<td>Muhammad Zam</td>
<td>8/13/99</td>
<td>C.Y.0.0001</td>
<td>8/4/99</td>
<td>Documentation</td>
<td>Functional</td>
</tr>
<tr>
<td>7/20/99</td>
<td>1</td>
<td>2</td>
<td>Chuck Chavall</td>
<td>7/30/99</td>
<td>C.Z.0.0001</td>
<td>8/2/99</td>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>7/24/99</td>
<td>4</td>
<td>3</td>
<td>Jenny Chung</td>
<td>8/12/99</td>
<td>D.Z.0.0001</td>
<td>8/5/99</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>7/24/99</td>
<td>5</td>
<td>4</td>
<td>Lary Kanemetsu</td>
<td>8/8/99</td>
<td>A.X.0.0001</td>
<td>8/2/99</td>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>7/26/99</td>
<td>1</td>
<td>5</td>
<td>Frank Currant</td>
<td>8/22/99</td>
<td>A.X.0.0001</td>
<td>8/7/99</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>7/29/99</td>
<td>5</td>
<td>4</td>
<td>Chuck Chavall</td>
<td>8/20/99</td>
<td>A.X.0.0001</td>
<td>8/11/99</td>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>7/31/99</td>
<td>1</td>
<td>3</td>
<td>Jenny Chung</td>
<td>8/5/99</td>
<td>A.X.0.0001</td>
<td>8/9/99</td>
<td>Functional</td>
<td></td>
</tr>
<tr>
<td>8/26/99</td>
<td>2</td>
<td>3</td>
<td>Lary Kanemetsu</td>
<td>8/17/99</td>
<td>B.Y.0.0001</td>
<td>8/8/99</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>8/27/99</td>
<td>1</td>
<td>1</td>
<td>Muhammad Zam</td>
<td>8/1/99</td>
<td>B.Y.0.0001</td>
<td>8/10/99</td>
<td>Process</td>
<td></td>
</tr>
<tr>
<td>8/30/99</td>
<td>2</td>
<td>2</td>
<td>Chuck Chavall</td>
<td>8/20/99</td>
<td>C.X.0.0001</td>
<td>8/6/99</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>8/26/99</td>
<td>4</td>
<td>4</td>
<td>Muhammad Zam</td>
<td>8/14/99</td>
<td>C.Z.0.0001</td>
<td>8/2/99</td>
<td>System</td>
<td></td>
</tr>
<tr>
<td>8/27/99</td>
<td>1</td>
<td>5</td>
<td>Chuck Chavall</td>
<td>8/12/99</td>
<td>D.Z.0.0001</td>
<td>8/4/99</td>
<td>Functional</td>
<td></td>
</tr>
</tbody>
</table>
Open/Closed Bug Tracking
General Testing Approach

- Create lists of potential problems and categorize them by area.

- Design repeatable tests, for proof of problems and problem resolutions.
Broad Categories for Errors

- Errors in interpreting requirements
- Errors in translating requirements into design, i.e. in programming
- Errors in implementing design
- Errors in the testing process itself
Program Behavior Views
(sets of behaviors taken over all inputs)

Specified (desired) behaviors

Observable behaviors (produced by program)
The Ideal

Specified (desired) behaviors

Observable behaviors
The ideal might not be fully realizable because

- Some aspects of a specification may be left arbitrary, unspecified,
- meaning either:
  - The specification is to be regarded as incomplete, or
  - Any behavior *consistent with* the specification will be accepted.
Testing asks questions:

Does a behavior occur?

Specified (desired) behaviors

Observable behaviors

Tested Behaviors
(i.e. behaviors tested for)
Specified (desired) behaviors

Observable behaviors

Specified, but not tested

Not specified, but tested

Tested Behaviors
Which regions of the diagram indicate errors?

Specified (desired) behaviors

Observable behaviors

Tested Behaviors
Which regions of the diagram indicate errors?

Specified (desired) behaviors

Observable behaviors

Specified and tested, but not observed

Tested Behaviors

Not specified but tested and observed
Black- vs. White Box
(recap)

- **Black-box** focuses on the specification: What is in the spec that the program doesn’t do?

- **White-box** focuses on the program: What does the program do that is not in the spec?

- Normally don’t rely on one or the other exclusively.
Black-Box Testing
Good Black-Box Test Plan

Specified (desired) behaviors

Observable behaviors

Tested, Specified Behaviors

(as large as is feasible)
Testing Example (3)

- A third reputable programmer has produced a self-contained program “triangle” that reads triples of numbers at a time and classifies them as to whether they are the lengths of the sides of some triangle, and if so, what kind.

- Negative side-length counts as a side with the length as absolute value of the specified length.

- The sides are in a specified range between 1E-150 and 1E150.
Testing Example (3)

- The possible outputs are:
  - “not a triangle”
  - “equilateral triangle”
  - “isosceles triangle”
  - “scalene triangle”
  - Either of the last two above preceded by “right”.
  - Or none of the above, with an indication that one or more of the inputs is out of range.

- “All classifications are based on native machine arithmetic”.
- Determine how you would test this program.
Black-Box Techniques

- Recall that “black box” means we do not get to see the code; we only have access to an installation of the product.

- Also called “functional testing” (vs. “structural testing”, which would be “white box”)

- Driven by requirements, use cases
Black-Box Techniques

- Equivalence Partitioning:
  - Use a small number of test equivalence classes, rather than a large number of individual test data points.
  
  - The actual tests are representatives of the equivalence classes.
  
  - Partitioning based on their relative likelihood of exposing logic errors in the code.
  
  - Example: Partition a number space into:
    - less than 0
    - equal to 0
    - greater than 0, less than 100
    - greater than or equal to 100
Black-Box Techniques

- Equivalence Partitioning Examples (cont’d):
  - Partition a **two-dimensional** number space into \((x, y)\) where:
    - \(x < y\)
    - \(x == y\)
    - \(x > y\)
  - Why these?

- Partition a String space into
  - length = 0
  - length = 1
  - length = 2
  - length = 3
  - length between 4 and 100
  - length greater than 100
Black-Box Techniques

- How would you equivalence-partition the triangle program input space?
Decision Table

A “declarative” means to categorize input

- Partitions input space (triples of numbers) into equivalence classes.
- All inputs in a cell should have the same anticipated equivalence class.

<table>
<thead>
<tr>
<th>Input Categories</th>
<th>a, b, c a triangle?</th>
<th>N</th>
<th>Y</th>
<th>Y</th>
<th>Y</th>
<th>N</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a = b?</td>
<td></td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>a = c?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = c?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Categories</td>
<td>not a triangle</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>scalene</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>isosceles</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>equilateral</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>should not occur</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Input Categories
- Output Categories
## Decision Table Variant

- Slightly more condensed, based on logical equivalences
- Eliminate or reduce “should not occur” entries

### Input Categories
- a, b, c a triangle?
- a = b?
- a = c?
- b = c?

### Output Categories
- not a triangle
- scalene
- isosceles
- equilateral

<table>
<thead>
<tr>
<th>N</th>
<th>Y</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>-</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>-</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Black-Box Techniques

- **Boundary-value testing**: Pick test cases near to “natural” boundaries in data space, so as to test whether the program performs the correct classification of borderline cases.
Black-Box Techniques

- **Logarithmic testing**: Repeatedly split the data space into two, testing sample points in the upper and lower halves of the split, then repeat on the lower half only.
Black-Box Techniques

- **Cause-Effect Graphing**: Examine requirements specification for logical **chains of conditions**; develop test cases that check whether these chains are actually observed.

  - Example: “If the clipboard is empty, then the paste menu option should **not** be selectable.”

  - Therefore: Develop a test in which the clipboard should be empty, and check that the menu option is not selectable.
Black-Box Techniques

- **Cause-Effect Graphing**, possible relationships:
  - A condition *implies* an action
  - A condition *precludes* an action
  - Two actions are *mutually* exclusive
  - A combination (conjunction) of two conditions implies an action
  - etc.
Example Cause-Effect Graph

Causes

- C1
- C2
- C3
- C4
- C5

Effects

- E1
- E2
- E3

Conjunctions, negations, etc.

(disjunction implied)
Black-Box Techniques

- **Comparison Testing:**

  Test product side-by-side with a “gold standard”, a program believed to be correctly operating (such as an earlier version having most, if not all, of the features)
Black-Box Techniques

- **Garbage-In Test**: See if unusual input characters, click sequences, etc. can force the system into inconsistent states.
- **Open-Book Test**
- **Data-Quantity Stress Test**: See if unusually large amounts of data cause nominal values to be exceeded, revealing untested overflow conditions, etc.