Software Testing

David Janzen

Verification and Validation

- Validation: is the system correct with respect to some specification?
- Verification: did we build the right system?
- V&V differences don't matter
- V&V generally refers to any activity that attempts to ensure that the software will function as required

V&V Activities

- Reviews, Inspections, and Walkthroughs
- Formal verification
- Testing
  - Formal and informal methods
  - Dynamic (run tests)
  - Levels: Unit, Integration, System, Regression
  - Techniques: Functional (black-box), Structural (white/clear-box), Stress, Usability, …

Testing

- A process of executing a program with the intent of finding errors
  - Def: The *dynamic* verification of the behavior of a program on a *finite* set of test cases, suitably *selected* from the usually infinite executions domain, against the *expected* behavior
- Objective: to find defects
- Can detect the presence of defects, but not their absence

Testing Glossary

- Error: mistake, bug
- Fault: result of an error, defect
- Failure: when a fault executes
- Incident: symptom associated with a failure
- Test Case: set of inputs and expected output
- Clean Tests: show something works
- Dirty Tests: show something doesn’t work

Testing Approaches

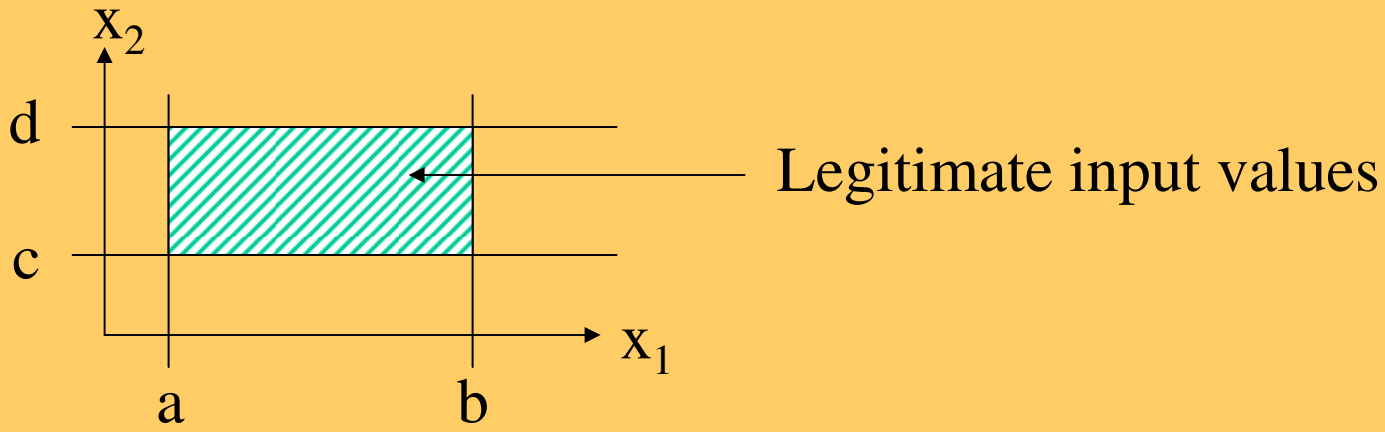
- Functional Testing (black-box)
  - Boundary Value Analysis
  - Equivalence Class
  - Decision Tables
  - Cause and Effect
- Structural Testing (white/clear-box)
  - Program graphs
  - Define-use paths
  - Program slicing

Equivalence Class Testing

- Partition input/output data into mutually disjoint sets where any number in the group is as good as another
  - Little league ages (8-12)
    - {(7 and lower) (8-12) (13 and higher)}
  - Months for number of days calculations
    - {(February)(30-day months)(31-day months)}
- Select test cases that involve values from all partitions

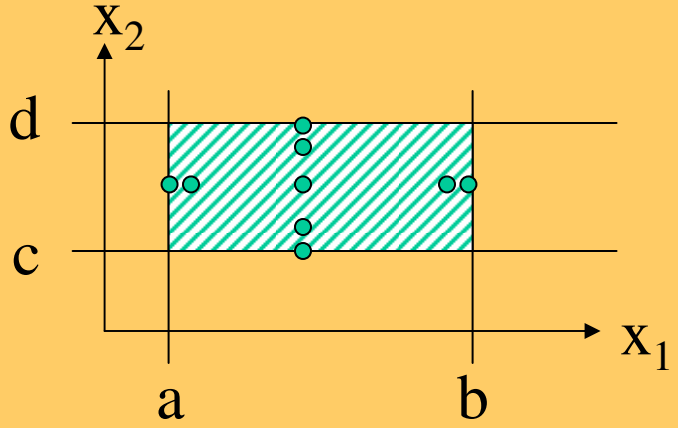
Boundary Value Analysis

- Think of a program as a function
  - $f(x_1, x_2)$
  - x_1 and x_2 have some boundaries
  - $a \leq x_1 \leq b$ (range of legitimate values)
  - $c \leq x_1 \leq d$ (a,b,c,d are boundary values)



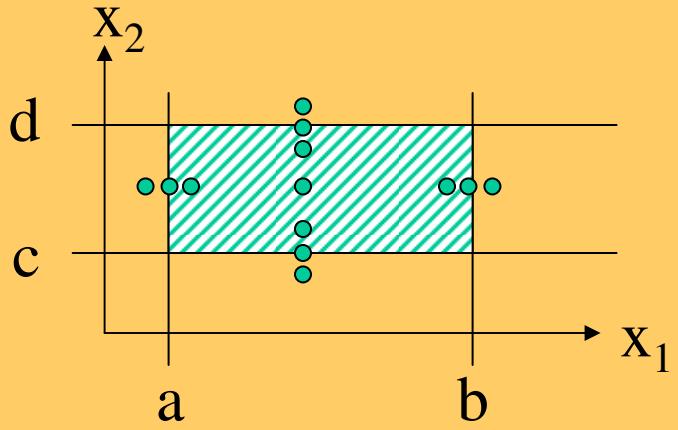
Boundary Value Analysis

- Premise: Bugs tend to lurk around the edges
- Single fault assumption
  - Hold all variables but one constant
  - Vary one to min, min+1, nominal, max-1, max
  - n variables yields 4n + 1 test cases



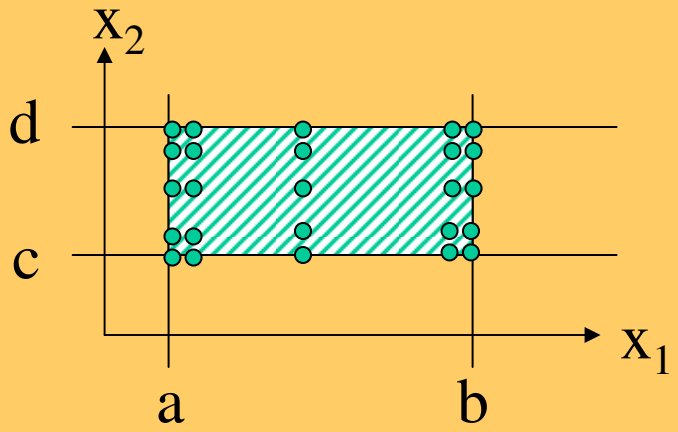
BVA Variation

- Also test beyond boundaries
  - min-1, max+1
  - n variables yields 6n + 1 test cases



Worst-case BVA

- Reject single fault assumption
  - Allow multiple variables to vary
  - n variables yields 5^n test cases



- Identify test cases that accomplish
  - Boundary Value Analysis testing (normal, variation, and worst-case)
  - Equivalence Class testing
  - 100% line, branch, and condition coverage

```
public boolean isIsosceles(int a, int b, int c) {
  if ((a < 1) || (b < 1) || (c < 1))
    return false;
  if ((a == b) || (a == c) || (b == c))
    return true;
  else
    return false;
}
```

Decision Tables

- Triangle example
  - Inputs: length of sides a, b, c
  - Outputs: type of triangle (equilateral, isosceles, scalene, not a triangle, impossible)

c1:a<b+c	F	T	T	T	T	T	T	T	T	T	T
c2:b<a+c		F	T	T	T	T	T	T	T	T	T
c3:c<a+b			F	T	T	T	T	T	T	T	T
c4:a=b				T	T	T	T	F	F	F	F
c5:a=c				T	T	F	F	T	T	F	F
c6:b=c				T	F	T	F	T	F	T	F
a1:Not a Triangle	X	X	X								
a2:Scalene											X
a3:Isosceles							X		X	X	
a4:Equilateral				X							
a5:Impossible					X	X		X			

Path Testing

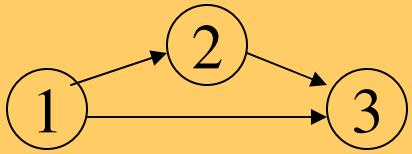
- Related to cyclomatic complexity
- Think of a module as a directed graph where nodes are statements or conditions
- Independent basis paths
  - Any path through the program that introduces at least a new set of statements or a new condition
- Write test cases that correspond to paths

Flow Graph Mappings

- Sequence

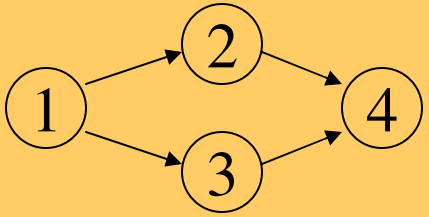

```
x = y + 5; //1
z = x / y;  //2
```

- Selection (if-then)




```
if(x>y) {//1
    z = x / y;  //2
}
y = z -2; //3
```

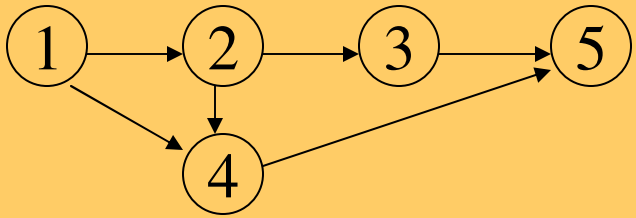
- Selection (if-then-else)




```
if(x>y) {//1
    z = x / y;  //2
} else {
    z = x * y; //3
}
y = z -2; //4
```

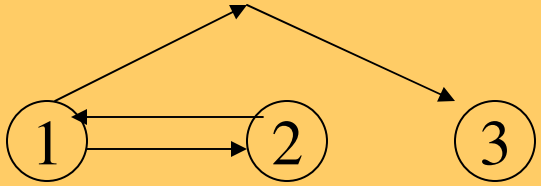
Flow Graph Mappings

- Selection (multiple condition if-then)




```
if((x>y) //1
    && (y<z)) {//2
    z = x / y;  //3
} else {
    z = x * y; //4
}
y = z -2; //5
```

- While




```
while(x>y) {//1
    z = x / y;  //2
}
y = z -2; //3
```

Program Slicing

- A form of data-flow testing
- A slice is the subset of a program that relates to a particular location
- Collect only code that “touches” variables used in computation at desired location
  - Simplifies testing
  - Can be done statically

Mutation Testing

- Also known as fault seeding
- Insert faults to see if test cases catch them
- Jester is a Java tool to do this

Test Adequacy

- How do we know when we are done testing?
  - We don't
  - When defect discovery rate is reasonably low
  - When test coverage is reasonably high
  - When defects found meets defects predicted
    - Size predictors (x defects per LOC expected)
    - Capture-Mark-Recapture (see next slide)
    - Bayesian Belief Networks

Capture-Mark-Recapture

- Two independent test teams
  - Team A detected N_A defects
  - Team B detected N_B defects
  - N_C represents defects found by both teams
- Estimate number of undiscovered defects
  - $(N_A * N_B)/ N_C – (N_A + N_B - N_C)$