Software Quality Assurance

David Janzen

What is quality?

- Crosby: “Conformance to requirements”
  - Issues:
    - who establishes requirements?
    - implicit requirements
- Juran: “Fitness for intended use”
  - Issues:
    - Who defines fitness? Novice users, experts, engineers?
- IEEE: “The degree to which the software possesses a desired combination of attributes”
  - Possible attributes:
    - usability, features, performance, 0 defects, low cost, elegant code, …

Quality Evolution 1

- Quality Control
  - Measure quality *after* system is built
  - Typical practices:
    - Testing, inspections, metrics at end of construction
    - E.g. # requirements met, # tests passed, coupling
  - Problems:
    - Have we tested enough?
    - Defect fixes inject new defects
    - Result in adversarial relationships

Quality Evolution 2

- Quality Assurance
  - IEEE: “A planned and systematic pattern of all actions necessary to provide adequate confidence that the product conforms to established technical requirements”
  - Typical practices:
    - Inspections, reviews, audits, metrics, communication throughout development process
    - SQA Plan (see examples on web)
  - Problems:
    - QA skills are rare
    - Separate QA team: communication issues, disputes
    - Commitment to QA wanes under schedule pressure

Quality Evolution 3

- Quality Engineering
  - Build quality as part of the SE process
  - Typical practices:
    - Everyone considers quality part of their job
    - Finding defects is good
    - QA team coaches/mentors, not evaluators
    - Fact-based decision-making based on metrics
  - Problems:
    - Process and cultural change

Quality is Free

- Crosby: “Quality is free. But it is not a gift.”
  - Prevent defects rather than remove them
  - “Zero-Defect is the attitude of defect prevention. It means, 'do the job right the first time.‘”

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

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

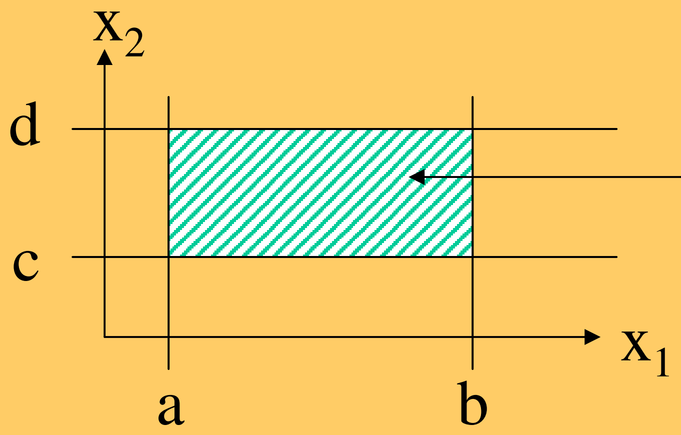
- A process of executing a program with the intent of finding errors
- Objective: to find defects
- Can detect the presence of defects, but not their absence

Testing Approaches

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

Boundary Value Analysis

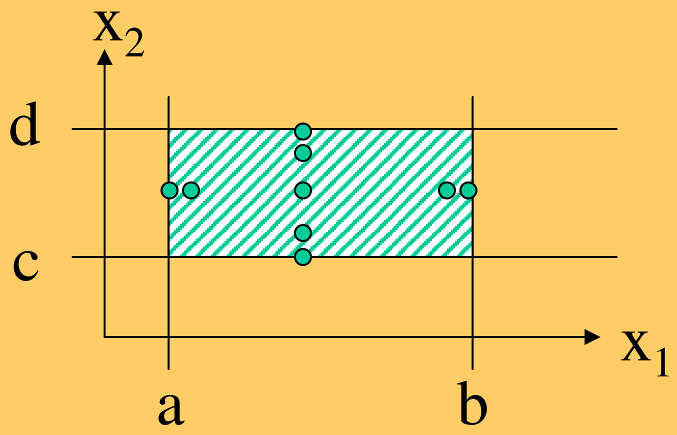
- Think of a program as a function
  - $f(x_1, x_2)$
  - x_1 and x_2 have some boundaries
  - $a \leq x_1 \leq b$ (range of legitimate values)
  - $c \leq x_2 \leq d$ (a,b,c,d are boundary values)



Legitimate input values

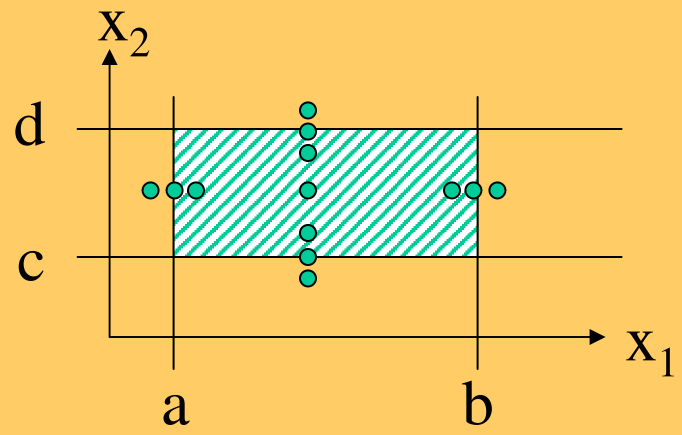
Boundary Value Analysis

- 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



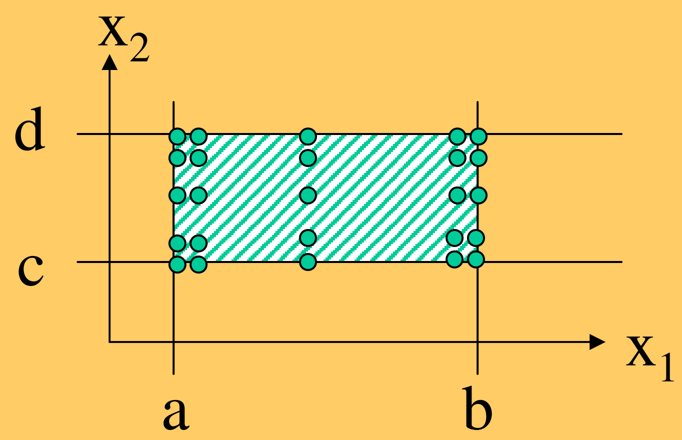
BVA Variation

- 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



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

- 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

- See reading

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

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
  - – $(N_A * N_B)/ N_C – (N_A + N_B - N_C)$