INF 117
Project in Software Engineering

Lecture Notes – Spring Quarter, 2008

Michele Rousseau

Set 8 – Testing

What's Next

Announcements

- Drop Boxes
  - We will use drop boxes for the remainder of the qtr
  - Please still post all deliverables
    - EXCEPT: Team Appraisals, Peer Evals & Course Logs
- Due: Thursday
  - Design Iteration #3
  - Code Iteration #1
  - Project Plan #3
- Friday: Cust Approval of Design

Today's Class

- Testing
  - Coverage-Based Testing
  - Equivalence Partitioning
  - Boundary Value Testing

Motivation

- People are not perfect
  - We make errors in design and code
  - Goal of testing: given some code, uncover as many errors are possible
- Important and expensive activity
  - Not unusual to spend 30-40% of total project effort on testing

The Purpose of Testing

Design and coding are creative. but…

- Testing is Destructive
  - The primary goal is to "break" the software
- Very often the same person does both coding and testing
  - This is not ideal… why?
  - Need "split personality":
    - when you start testing, become paranoid and malicious
  - Surprisingly hard to do: people don't like finding out that they made mistakes

Testing Approach

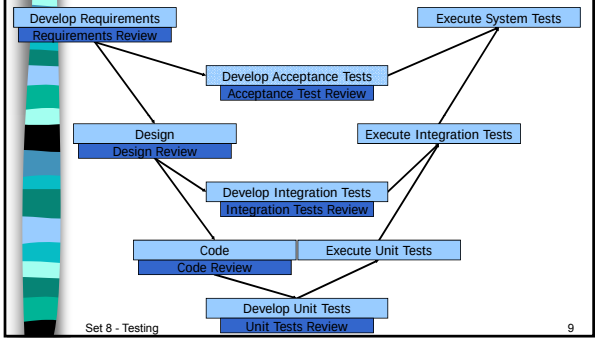
Testing is a process of executing software with the intent of finding errors

- Good testing has a high probability of finding as-yet-undiscovered errors
- Successful testing discovers unknown errors
- If did not find any errors, need to ask "Was our testing approach is good?"

Testing

- Unit Testing
- Integration Testing
- System Testing
- Regression Testing

V-Model of Development & Testing (the big picture)



Fundamental Testing Questions

- Test Criteria: What should we test?
- Test Oracle: Is the test correct?
- Test Adequacy: How much is enough?
- Test Process: Is our testing effective?

How to make the most of limited resources?

Some Commonly Used Testing Approaches

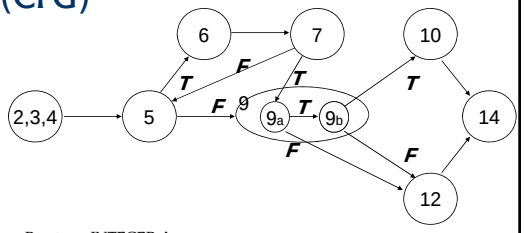
- Coverage or Control-flow based
- Data-flow based
- Equivalence Partitioning
- Boundary Value Analysis

Coverage-Based Testing

- Flow Graphs
  - Control Flow
    - Partial order of Statement Execution
- Data Flow
  - Flow of values from Definition to Variables

Graph representation of control flow and data flow relationships

Prog P's Control Flow Graph (CFG)



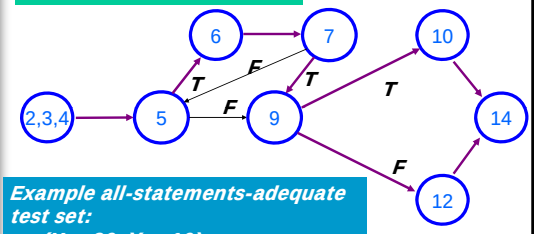

```
1  function P return INTEGER is
2  begin
3      X, Y: INTEGER;
4      READ(X); READ(Y);
5      while (X > 10) loop
6          X := X – 10;
7          exit when X = 10;
8      end loop;
9      if (Y < 20 and then X mod 2 = 0) then
10         Y := Y + 20;
11     else
12         Y := Y – 20;
13     end if;
14     return 2*X + Y;
15 end P;
```


All-Statements Coverage

- Select test cases such that every node in the graph is visited
  - Also called node coverage
    - Guarantees that every statement in the source code is executed at least once
- Selects minimal number of test cases

All-Statements Coverage of P

At least 2 test cases needed



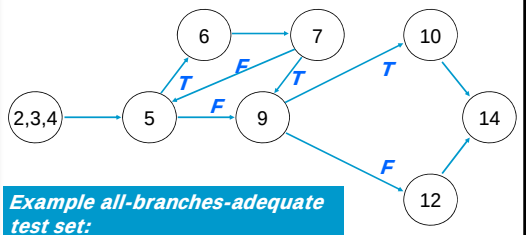
Example all-statements-adequate test set:
(X = 20, Y = 10)
<2,3,4,5,6,7,9,10,14>
(X = 20, Y = 30)
<2,3,4,5,6,7,9,12,14>

All-Branches Coverage

- Select test cases such that every branch in the graph is visited
  - Guarantees that every branch in the source code is executed at least once
- More thorough than All-Statements coverage
  - More likely to reveal logical errors

All-Branches Coverage of P

*At **least** 2 test cases needed*



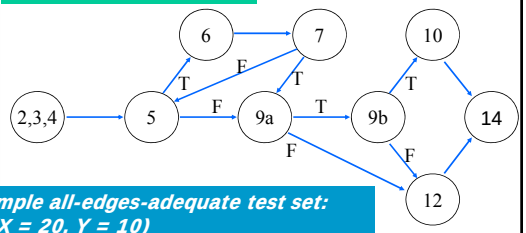
Example all-branches-adequate test set:
(X = 20, Y = 10)
<2,3,4,5,6,7,9,10,14>
(X = 15, Y = 30)
<2,3,4,5,6,7,5,9,12,14>

All-Edges Coverage

- Select test cases such that every edge in the graph is visited
  - Takes complex statements into consideration – treats them as separate nodes
- More thorough than All-Branches coverage
  - More likely to reveal logical errors

All-Edges Coverage of P

At least 3 test cases needed



Example all-edges-adequate test set:
(X = 20, Y = 10)
<2,3,4,5,6,7,9a,9b,10,14>
(X = 5, Y = 30)
<2,3,4,5,9a,12,14>
(X = 21, Y = 10)
<2,3,4,5,6,7,5,6,7,5,9a,9b,12,14>

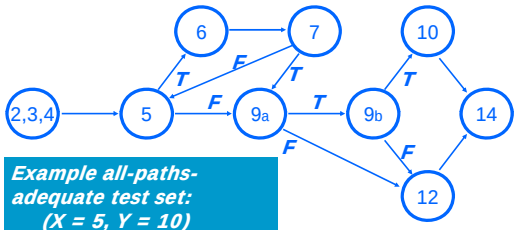
All-Paths Coverage

- Path coverage
  - Select test cases such that every path in the graph is visited
  - Loops are a problem
    - 0, 1, average, max iterations
- Most thorough...

...but is it feasible?

All-Paths Coverage of P

Infinitely many test cases needed



Example all-paths-adequate test set:
(X = 5, Y = 10)
(X = 15, Y = 10)
(X = 25, Y = 10)
(X = 35, Y = 10)
...

Subsumption of Criteria

- C1 subsumes C2 if any C1-adequate test T is also C2-adequate
  - But some T1 satisfying C1 may detect fewer faults than some T2 satisfying C2

Data-flow

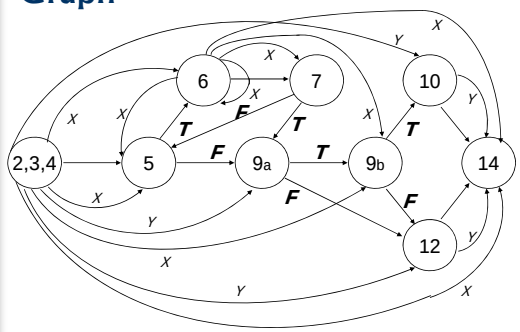
Test connections between variable definitions ("write") and variable uses ("read")

- Variation of the control flow graph
  - A node represents a single statement, not a single-entry-single-exit chain of statements
- Set **DEF(n)** contains variables that are defined at node n (i.e., they are written)
- Set **USE(n)**: variables that are read

Def-Use Pair

- A def-use (DU) pair for variable **x** is a
- pair of nodes **(n1,n2)** such that
  - **x** is in **DEF(n1)**
  - the definition of **x** at **n1** reaches **n2**
  - **x** is in **USE(n2)**
- i.e., the value that is assigned to **x** at **n1** is used at **n2**
  - Since the definition reaches **n2**, the value is not "killed" along some path **n1...n2**

P's Control and Data Flow Graph



Test Adequacy

Tells you when to stop testing

- Coverage-Based Testing
  - Coverage metrics
    - when sufficient percentage of the program structure has been exercised
- Fault-Based Testing
  - Empirical assurance
    - when failures/test curve flatten out
  - Error seeding
    - percentage of seeded faults found is proportional to the percentage of real faults found
- Error-Based Testing
  - faults found in common are representative of total population of faults
  - Equivalence Partitioning

Test Criteria

- Testing must select a subset of test cases that are likely to reveal failures
- Test Criteria provide the guidelines, rules, strategy by which test cases are selected
  - actual test data
  - conditions on test data
  - requirements on test data
- Equivalence partitioning is the typical approach
  - a test of any value in a given class is equivalent to a test of any other value in that class
  - if a test case in a class reveals a failure, then any other test case in that class should reveal the failure
  - some approaches limit conclusions to some chosen class of errors and/or failures

Equivalence Partitioning (EQP)

- **Testing technique**
  - Reduces the # of test cases
    - Make the # of test cases manageable
    - Systematic derivation of test cases
  - Reasonably tests the system

Basic Principle:
Some distinctions don't make a difference

EQP : How does it work

Notice when any element in the partition will produce the same results

Divide inputs into *equivalent* partitions

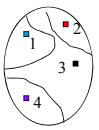
- i.e. Find a small # set of *representative* input values
- For each Class program behaves in an "equivalent" way
- Smaller test set – but equally effective

EQP: Reduces test cases

Input domain

Large set of test inputs.

Input domain partitioned into four sub-domains.

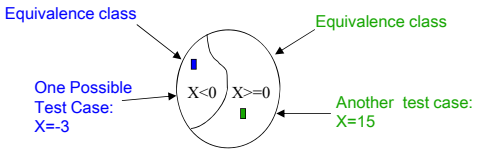


Four test inputs, one selected from each sub-domain.

How to partition? Example 1

- Suppose that program P takes an input X, X being an integer.
- For X<0 perform task (T1)
- For X>=0 perform task (T2)

Two sub-domains

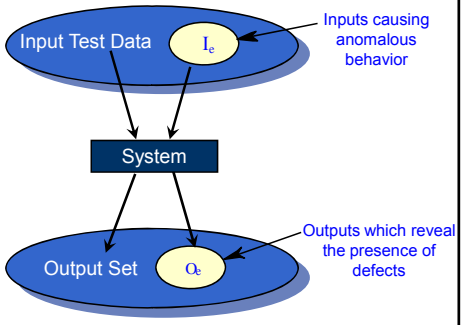


- **All test inputs in the X<0 sub-domain are considered equivalent.**
- **The assumption is that if one test input in this sub-domain reveals an error in the program, so will the others.**
- **This is true of the test inputs in the X>=0 sub-domain also.**

EQP: Basic Process

- First you must break the input into sub-domains (partitions)
  - Look at input and determine common properties
  - Values with in defined range
  - Values outside of the defined range
  - Extremes
  - Try to include input that will force incorrect output
    - How well does the code perform exception handling
- If the sub-domains are well done
  - should be able to create a few (or ideally) one test case that will represent the entire domain

Include inputs in and out of range



EQP: Example 2

- Input should be a numerical month
  - Valid Inputs: 1-12
- What are potential Classes?
  - Input within range:
    - 1-12
  - Out of Range
    - High End: 20, 99, 3-digit, 4-digit
    - Low End: Negative Numbers
    - Alphanumeric
    - Special Characters / Punctuation

Boundary Value Analysis (BVA)

- Select test cases based on the boundaries values
- Look for inputs
  - On the boundary
  - On either side of the boundary
- For numeric month example
  - Boundary Values
    - Low End: 0,1,2
    - High End: 11,12,13
- Combining this technique with Equivalence Partitioning is much more effective

EQP &BVA

- Input
  - 5-digit integer between 10,000 and 99,999,
- Partitions
  - <10,000
  - 10,000-99,999
  - > 10, 000
- Boundary Values
  - 00000
  - 09,999 –10,000
  - 99,998 – 99,999 – 100,000
- Outside
  - Alphanumeric
  - Symbols

What do you need to do?

- Have a plan!
  - Not monkey testing
- Create test cases wisely (think about what they are covering)
- Define your test cases and results
- See Read-set templates

Ready-set Templates

- http://readyset.tigris.org/nonav/templates/frameset.html
- http://readyset.tigris.org/nonav/templates/frameset.html
- Add results.. Pass/fail

Integration Testing

- Purpose: to exercise the interfaces between classes/modules
  - Driven by design
  - What should it take in?
  - What should it supply?
  - What happens if they send the wrong stuff?
- Basic approaches
  - Top-Down
  - Bottom-up

Run tests developed during the design phase

Which Approach to use?

- Top-Down or Bottom Up?
- In practice, most integration involves a combination of these strategies

System Testing

- Black box...
- Running Acceptance Test Plan

One Last Announcement

- No class Wednesday