INF 111 / CSE 121: Software Tools and Methods

Lecture Notes for Fall Quarter, 2007
Michele Rousseau

Set 5 - Testing
(Some slides adapted from Sommerville 2000 & Scott Miller)

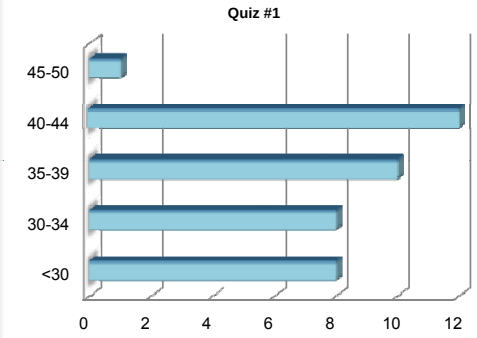
Announcements

- **Assignment #2 – Will be available this week – Due on July 21st**
- **Quiz #1 – Regrades need to be turned in by Thursday (7/10)**
  - Will regrades be accepted after Thursday?
- **Quiz #2 – Thursday (7/10)**
  - WILL INCLUDE:
    - All readings assigned since the last quiz not in the "WILL NOT INCLUDE LIST"
    - Slide set 4 & 5
  - WILL NOT INCLUDE:
    - Ch 2 from "The Mythical Man-Month"
    - Van Vliet Ch. 4 & 10 will not be included on this quiz

Quiz Results

- **Max → 48**
- **Min →13**
- **Median 36**
  - Not including the student s who did not take it

Breakdown of grades



Some perspective

Quiz is worth 5% of your grade
If I miss one can I still get an A?

I suffer from test anxiety – what can I do?

http://www.studygs.net/tstprp8.htm
http://ub-counseling.buffalo.edu/stresstestanxiety.shtml
http://www.sdc.uwo.ca/learning/mcanx.html
http://www.kidshealth.org/teen/school_jobs/school/test_anxiety.html

How do I improve my performance on Quizzes – and the final?

- **If you have to miss lecture –**
  - get notes from your friends
- **Review lecture slides (take notes)**
- **Do the reading**
- **Attend discussion section**
- **Form a study group**
- **Ask questions**
  - In class
  - Email
  - Office hours
- **What if I aced it? → WTG!**

Previous on INF 111…

- **Quiz #1**
- **More on Testing**
  - Static Analysis
    - Code Walkthroughs
    - Inspections
  - Issues in Quality Assurance

Today's Lecture

- **More on Testing**
  - Formal Methods
  - Fundamental Testing Questions
  - Test Adequacy
    - Coverage Based Testing

Issues with Testing own code

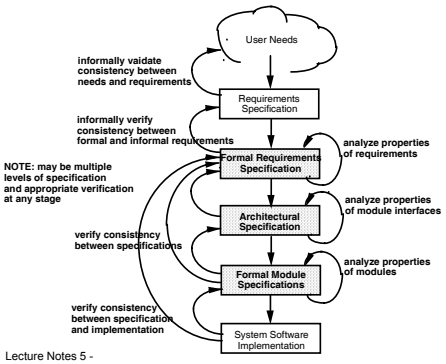
- **Developer tests the code just produced**
  - Needs to ensure that the code functions properly before releasing it to the other developers
- **Benefits**
  - Knows the code best
  - Has easy access to the code
- **Drawbacks**
  - Bias
    - "I trust my code"
    - "I always write correct code"
  - Blind spots
- **Possible Solutions:**
  - Outside Testers
  - Walkthroughs / Inspections

Formal Verification

- **Techniques for proving consistency between two software descriptions**
  - to prove consistency of specification
  - to prove correctness of implementation

Correctness →
Correct with respect to the specification

Verification with Formal Specs



Formal Verification / Validation

- **Some shortcomings**
  - does not show other qualities
    - Performace, usability, etc..
  - May not scale up
  - only informal techniques for validating against user needs
  - subject to assumptions of proof system
  - only as good as formal specification
  - Not trivial → tedious
  - Not always cost effective
- **Generally used on a part of the system**
- **Example: Mathematically Based Verification**

Mathematically Based Verification

- **Must have formal specifications**
  - Notation must be consistent with mathematical verification techniques
- **The programming lang. must have formal semantics**
- **This is an intensive process but…**
  - Can verify correctness
- **Generally,**
  - Not cost effective for large systems

Tools for Mathematical Verification

- **Can it be automated?**
  - Theorem provers
    - Assist in developing proofs
  - Usually work with a subset of the program
  - Not completely automated

The problem with Testing

- **Can't test exhaustively**
  - **Not feasible to run all those test cases**
  - **Not feasible to validate them once they are run**
- **Want to verify software →**
- **Need to test →**

So,

- **Need to decide on test cases →**

But,

no set of test cases guarantees absence of bugs,

Testing Techniques

So,

- We need to find a ***systematic approach*** to selecting of test cases **that will lead to**:
  - **accurate,**
  - **acceptably thorough,**
  - **repeatable identification of errors, faults, and failures?**

Practical Issues

- **Purpose of testing**
  - Fault detection
  - High assurance of reliability
  - Performance/stress/load
  - Regression testing of new versions
- **Conflicting considerations**
  - safety, liability, risk, customer satisfaction, resources, schedule, market windows and share
- **Test Selection is a sampling technique**
  - choose a finite set from an infinite domain

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?

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

Test Oracles

- **Where does "expected output" come from?**

*A **test oracle** is a mechanism for deciding whether a test case execution failed or succeeded*

- **Critical to testing**
- **Difficult to create systematically**
- **Typically done with a lot of guesswork**
  - Typically relies on humans
  - great dependence on the intuition of testers
- **Formal specifications make it possible to automate oracles**

What Does an Oracle Do?

- **Your test shows cos(0.5) = 0.8775825619**
- **You have to decide whether this answer is correct?**
- **You need an oracle**
  - Draw a triangle and measure the sides
  - Look up cosine of 0.5 in a book
  - Compute the value using Taylor series expansion
  - Check the answer with your desk calculator

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**
  - Independent testing
    - faults found in common are representative of total population of faults

Before the Break

- **Testing**
  - Formal Methods
  - Fundamental Testing Questions

Coverage-Based Testing

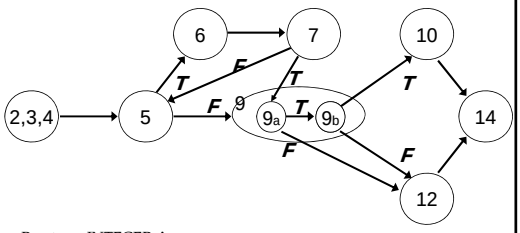
- **Flow Graphs**
  - Control Flow
    - Partial order of Statement Execution
  - Data Flow
    - Flow of values from Definition to Variables

Graph representation of control flow and data flow relationships

A Sample Program to Test

```
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;
```

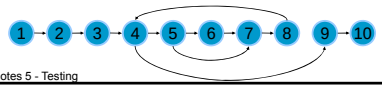

Prog P's Control Flow Graph (CFG)




```
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;
```

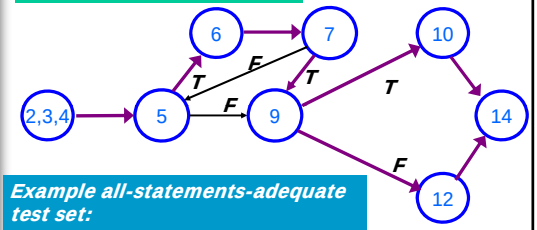

All-Statements Coverage

- **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

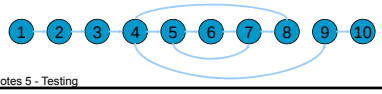


Example all-statements-adequate test set:

(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>

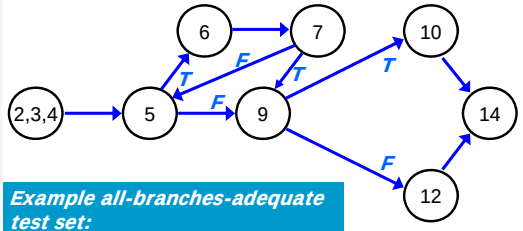
All-Branches Coverage

- **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:

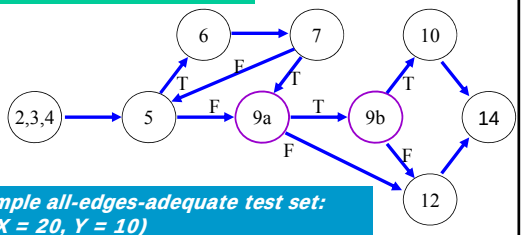
(X = 20, Y = 10)
<2,3,4,5,6,7,9,10,14>
(X = 15, Y = 30)
<2,3,4,5,6,7,5,9,12,14>

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 more complex 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>

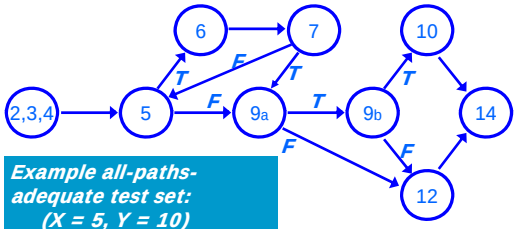
All-Paths Coverage

- **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?

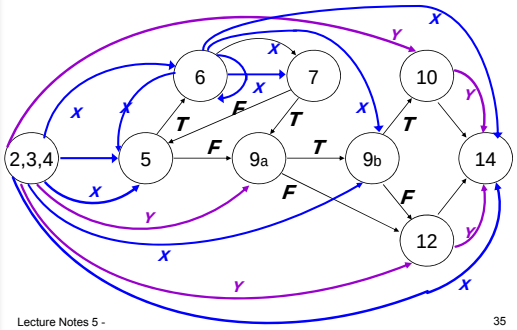
All-Paths Coverage of P

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)
…

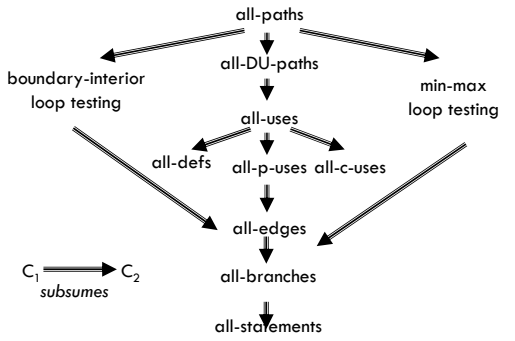
P's Control and Data Flow Graph



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

Structural Subsumption Hierarchy



Subsumes => Stronger than

Moving on.

- **Equivalence Partitioning & Boundary Value Analysis**
- **Integration Testing**
  - Top-Down
  - Bottom Up

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

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

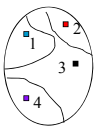
Basic Method:
Notice when any element in the partition will produce the same results (ie find the same faults)

EQP: Reduces test cases

Input domain

Large set of test inputs.

Input domain partitioned into four sub-domains.

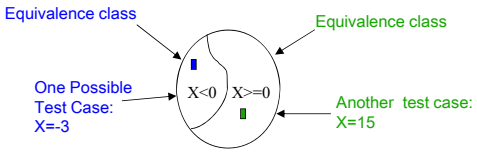


Four test inputs, one selected from each sub-domain.

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

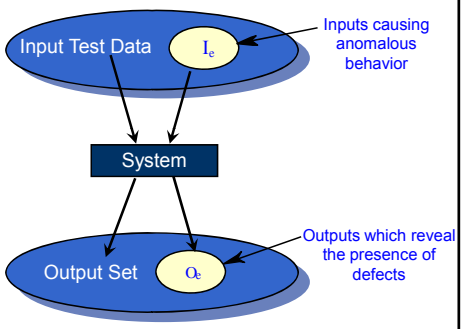


- **All test inputs in the X<0 sub-domain are considered equivalent.**
- **The assumption is that if one test input in this sub-domain reveals an error in the program, so will the others.**
- **This is true of the test inputs in the X>=0 sub-domain also.**

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)

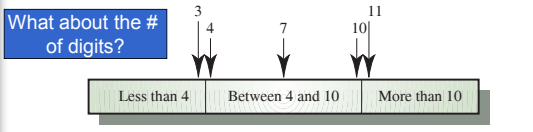
- **Select test cases based on the boundaries values**
- **Look for inputs**
  - On the boundary
  - On either side of the boundary
- **For numeric month example**
  - Boundary Values
    - Low End: 0,1,2
    - High End: 11,12,13
- **Combining this technique with Equivalence Partitioning is much more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

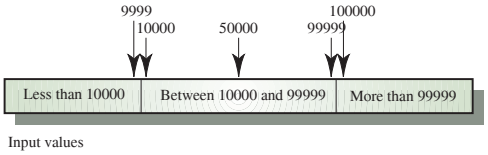
EQP & BVA

- **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

Equivalence partitions

What about the # of digits?





Pros & Cons

- **Pros**
  - Minimizes test cases & maintains some test adequacy
  - Forces tester to analyze the input and output domain
- **Cons**
  - Assumptions of equivalence can be tricky – and incorrect
  - Doing EQP is easy → Doing it well is difficult
  - Somewhat subjective – dependent upon the testers' intuition

Picking the correct subdomain can be tricky

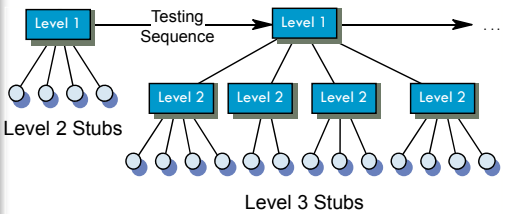
Today's Lecture

- **Equivalence Partitioning & Boundary Value Analysis**
- **Integration Testing**
  - Top-Down
  - Bottom Up

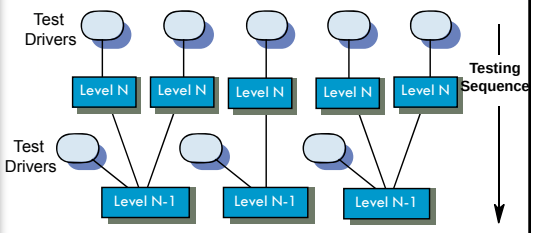
Integration Testing Approaches

- **Top Down & Bottom Up**
  - Top-down integration testing →
    - better at discovering errors in the system architecture
    - allows a limited demonstration at an early stage in the development
  - Bottom up →
    - Often easier to implement
- **Problems with both approaches. Extra code may be required to observe tests**

Top-Down Integration Testing



Bottom-Up Testing



Which Approach to use?

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