

# INF 111 / CSE 121: Software Tools and Methods

Lecture Notes for Fall Quarter, 2007  
Michele Rousseau  
Set 10

---

---

---

---

---

---

---

---

## Announcements

- o Checkmate
- o Quiz #1 – Will be returned on Wed
- o Quiz #2 – Next Monday 10/29/07

Topic 10

2

---

---

---

---

---

---

---

---

## Previous Lecture

- o Testing
  - Static Analysis
    - ▣ Code Walkthroughs
    - ▣ Inspections

Topic 10

3

---

---

---


---

---

---

---

---



## Today's Lecture

- **More on Testing**
  - Static Analysis
  - Formal Verification
  - Coverage-Based Testing

Topic 10 4

---

---

---


---

---

---

---

---



## Verification & Validation (revisited)

- **Verification**

*"Are we building the product right?" (Boehm)*

  - The Software should conform to its specification
  - testing, reviews, walk-throughs, inspections
  - internal consistency; consistency with previous step
- **Validation**

*"Are we building the right product?"*

  - The software should do what the user really requires
  - ascertaining software meets customer's intent

Topic 10 5

---

---

---


---

---

---

---

---



## Quality Assurance : 5 Problems

- **#1 : Eliciting the Customer's Intent**
  - Getting the Specs to meet the "real needs"
- **#2 : QA is inherently difficult**
  - Systems can be complex making QA difficult to perform
    - Air Traffic Control → stringent performance
    - Medical Diagnosis System → Complex processing

Topic 10 6

---

---

---

---

---

---

---

---

## Quality Assurance : 5 Problems

### #3 : Management Aspects

- Who does what testing?
  - Are developers involved?
- How are bugs handled?
- What is the reward structure?

### #4 : QA Team vs. Developers

- QA lays out the rules
- Uncovers faults
  - "image of competition"
- Viewed by Developers as Cumbersome
  - "let me just code"

### #5 : Can't test exhaustively

Topic 10

7

---

---

---

---

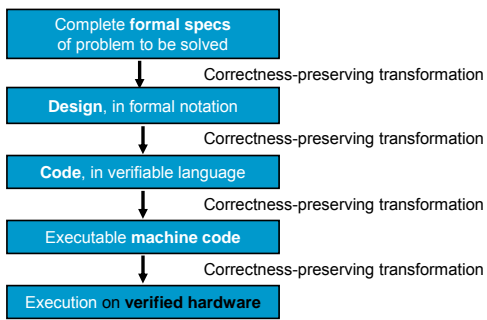
---

---

---

---

## How QA would like the world to be



Topic 10

8

---

---

---

---

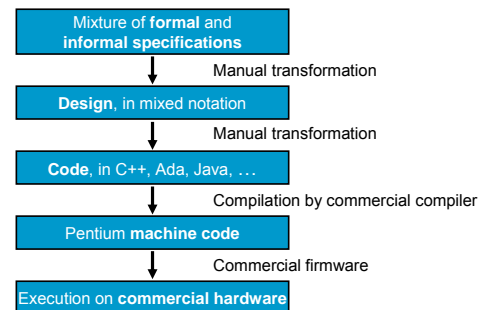
---

---

---

---

## ... but in reality



Topic 10

9

---

---

---

---

---

---

---

---

## Unit Tests

- **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

Topic 10

10

---

---

---

---

---

---

---

---

## 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*

Topic 10

11

---

---

---

---

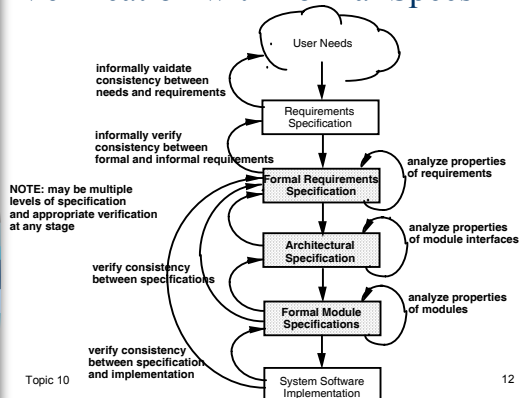
---

---

---

---

## Verification with Formal Specs



---

---

---

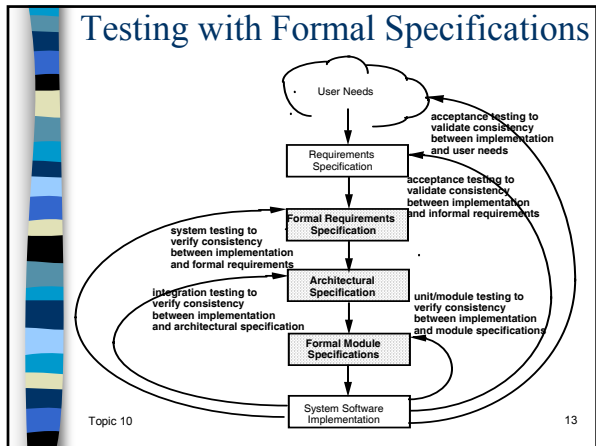
---

---

---

---

---




---

---

---

---

---

---

---

---

- ## Formal Verification / Validation
- **Some shortcomings**
    - does not show other qualities
      - Performance, 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**
- Topic 10 14

---

---

---

---

---

---

---

---

- ## 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
- Topic 10 15

---

---

---


---

---

---

---

---



## 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

Topic 10 16

---

---

---


---

---

---

---

---



## 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?

Topic 10 17

---

---

---


---

---

---

---

---



## 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

Topic 10 18

---

---

---

---

---

---

---

---

## 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?*

Topic 10

19

---

---

---

---

---

---

---

---

## 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

Topic 10

20

---

---

---

---

---

---

---

---

## 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**

Topic 10

21

---

---

---

---

---

---

---

---

## 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

Topic 10

22

---

---

---

---

---

---

---

---

## 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

Topic 10

23

---

---

---

---

---

---

---

---

## 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*

Topic 10

24

---

---

---

---

---

---

---

---



## 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;
```

Topic 10

25

---

---

---

---

---

---

---

---

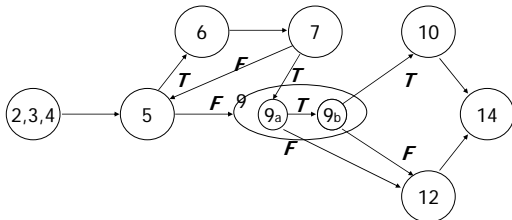
---

---

---

---

## 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;
```

26

---

---

---

---

---

---

---

---

---

---

---

---

## 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



Topic 10

27

---

---

---

---

---

---

---

---

---

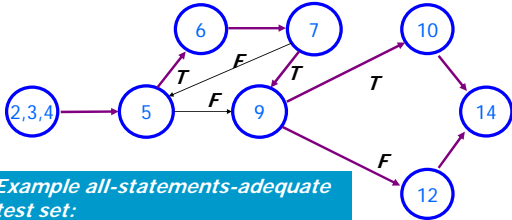
---

---

---

## All-Statements Coverage of P

At least 2 test cases needed



Example all-statements-adequate test set:

$(X = 20, Y = 10)$   
 $\langle 2, 3, 4, 5, 6, 7, 9, 10, 14 \rangle$   
 $(X = 20, Y = 30)$   
 $\langle 2, 3, 4, 5, 6, 7, 9, 12, 14 \rangle$

28

---

---

---

---

---

---

---

---

---

---

---

---

## 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



Topic 10

29

---

---

---

---

---

---

---

---

---

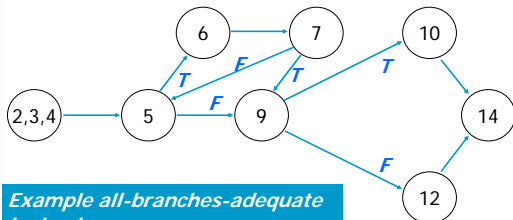
---

---

---

## All-Branches Coverage of P

At least 2 test cases needed



Example all-branches-adequate test set:

$(X = 20, Y = 10)$   
 $\langle 2, 3, 4, 5, 6, 7, 9, 10, 14 \rangle$   
 $(X = 15, Y = 30)$   
 $\langle 2, 3, 4, 5, 6, 7, 5, 9, 12, 14 \rangle$

30

---

---

---

---

---

---

---

---

---

---

---

---

## 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

Topic 10

31

---

---

---

---

---

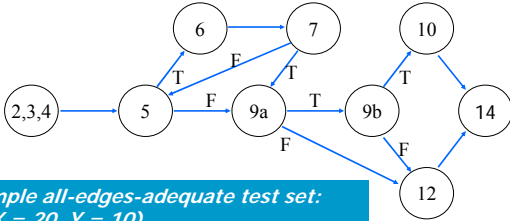
---

---

---

## 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>

32

---

---

---

---

---

---

---

---

## 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?

Topic 10

33

---

---

---

---

---

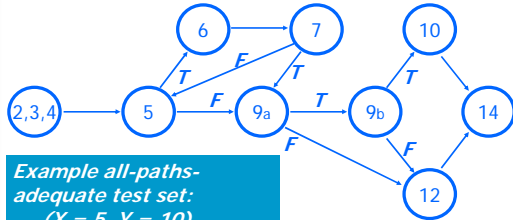
---

---

---

## 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)

...

34

---

---

---

---

---

---

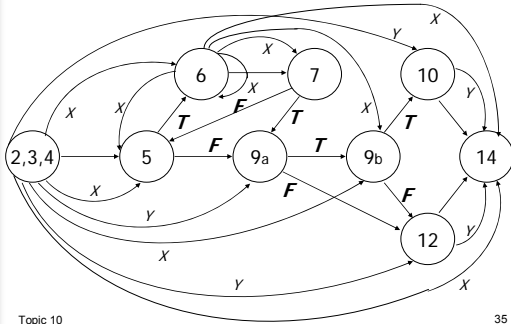
---

---

---

---

## P's Control and Data Flow Graph



Topic 10

35

---

---

---

---

---

---

---

---

---

---

## 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

Topic 10

36

---

---

---

---

---

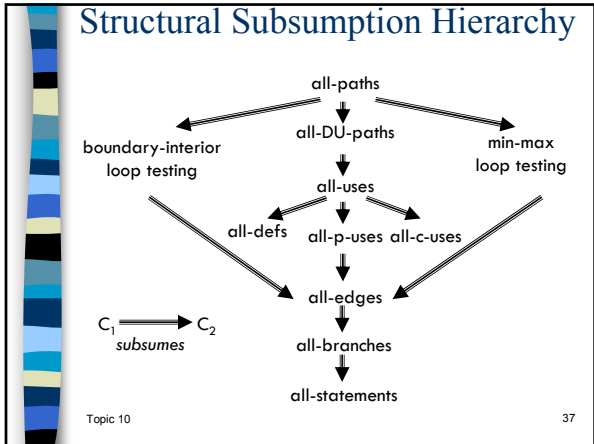
---

---

---

---

---




---



---



---



---



---



---



---