CS301 Software Development

Instructor: Peter Kemper

Coverage Criteria for Software Testing

Software Testing

Testing is never complete, so when is enough?

- Obviously:
  - If significant parts of program structure are not tested, testing is surely inadequate.
- Leads to idea of "Coverage"
- Statement coverage:
  - Test executes SUT code, let's monitor which lines are executed, which not.
  - Leads to:
    - Coverage % value for methods, classes, packages.
    - Color highlighting as a visualization in code editor.

Measuring Statement Coverage in Eclipse with Eclemma

- Eclemma (www.eclemma.org)
  - free Java code coverage tool for Eclipse
  - after installation, simply select under "coverage as" or 
  - produces statistics & highlights code

```
66          int xCoord = this.zeroLocation % this.dimensions;
67          int yCoord = (int)(this.zeroLocation / this.dimensions);
68          int[] newStateArray = this.getStateArray();
69          newStateArray = gamestate.clone();
70          int tempZeroLocation;
71
72          if(0==yCoord)
73          {
74              //Bad move
75
76              if (LOG.isLoggable(Level.INFO))
77              {
78                  LOG.info("Bad Move: MoveUp");
79              }
80
81              return null;
82          }
83          else
84          {
85              int temp = newStateArray[xCoord+(dimensions*(yCoord-1))];
86              newStateArray[xCoord+(dimensions*(yCoord-1))] = 0;
```

Measuring Statement Coverage in Eclipse with Eclemma

- Eclemma (www.eclemma.org)
  - free Java code coverage tool for Eclipse
  - after installation, simply select under "coverage as" or 
  - produces statistics & highlights code

Problems | Javadoc | Declaration | Console | Coverage | Call Hierarchy | SVN Repositories

AllTests (Sep 7, 2014 12:29:10 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
SlidingPuzzle	65.9 %	5,279	2,737	8,016
test	73.3 %	4,376	1,598	5,974
edu.wm.cs.cs301.slidingpuzzle	73.3 %	4,376	1,598	5,974
JUnitTestAdvanced.java	0.0 %	0	1,276	1,276
PuzzleGenerator.java	0.0 %	0	220	220
PuzzleSolverComparison.java	96.9 %	1,483	47	1,530
StateAndSolverTest.java	97.9 %	1,391	30	1,421
SimplePuzzleSolverTest.java	98.5 %	1,477	22	1,499
AllTests.java	0.0 %	0	3	3
FastPuzzleSolverTest.java	100.0 %	25	0	25
src	44.2 %	903	1,139	2,042
edu.wm.cs.cs301.slidingpuzzle	44.2 %	903	1,139	2,042
SlidePuzzleGUI.java	0.0 %	0	1,060	1,060
PanelType.java	0.0 %	0	60	60
SimplePuzzleState.java	97.7 %	503	12	515
Operation.java	92.9 %	65	5	70
SimplePuzzleSolver.java	99.4 %	328	2	330
FastPuzzleSolver.java	100.0 %	7	0	7

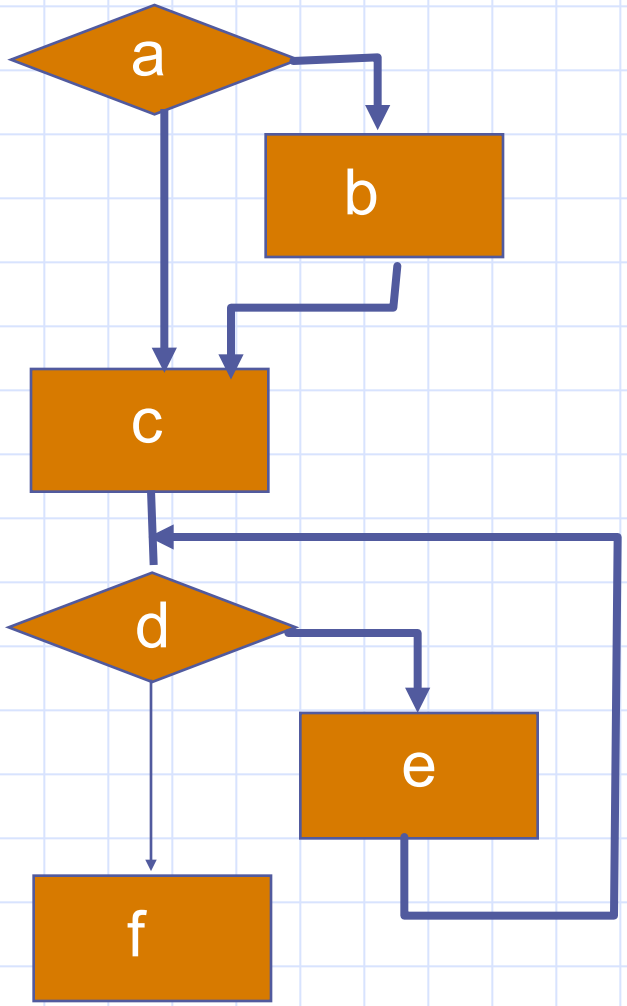
Various Control Flow Coverage Criteria Exist:

1. Statement (node, basic block) coverage
   - Node, single statement
   - Basic block (several statements in sequence)
2. Branch (edge) coverage
   - Branch: each branch of if/switch conditions
3. Condition coverage
   - Each basic condition evaluated to true and false, (A && B && C)
4. Path coverage (structured basis or cyclomatic testing)
   - Execution paths through a method
5. Data flow (syntactic dependency) coverage
   - Define-used pairs of data
6. Function coverage

… and there are more

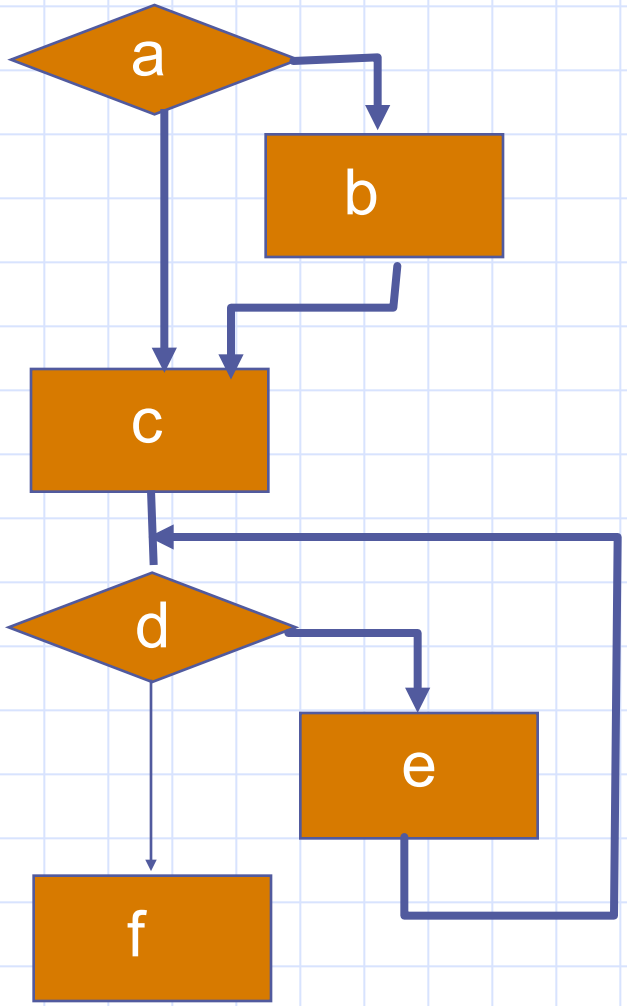
- Coverage typically measured as percentage

Control Flow Coverage Criteria: Statement, Branch, Path



Statement coverage covers all nodes (lines of code).

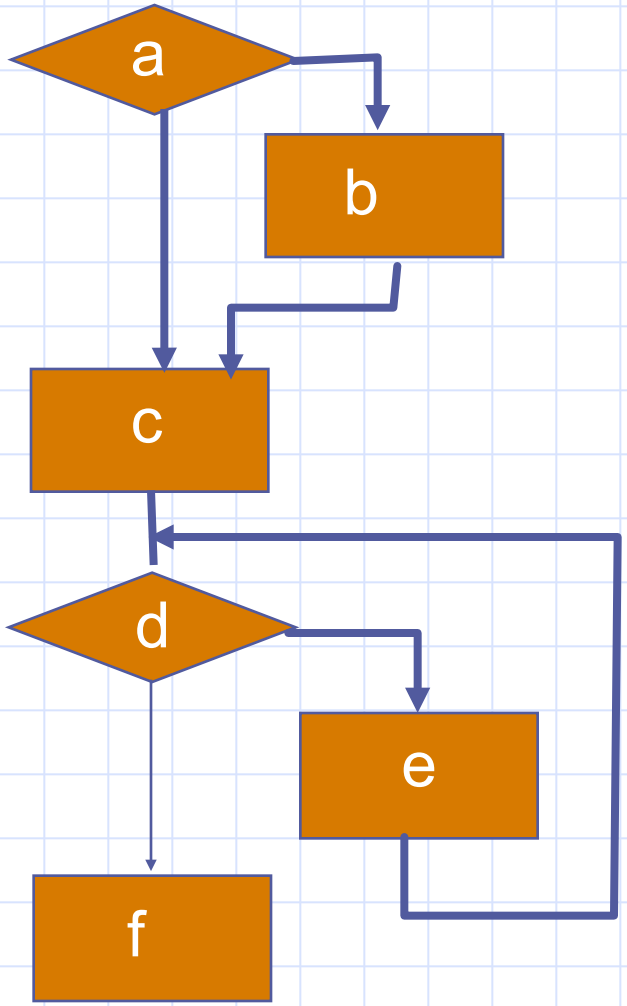
Control Flow Coverage Criteria: Statement, Branch, Path



We could also ask to cover
- all branches
- all paths, subpaths, …

Edge ac is required by branch but not by statement coverage

Control Flow Coverage Criteria: Statement, Branch, Path



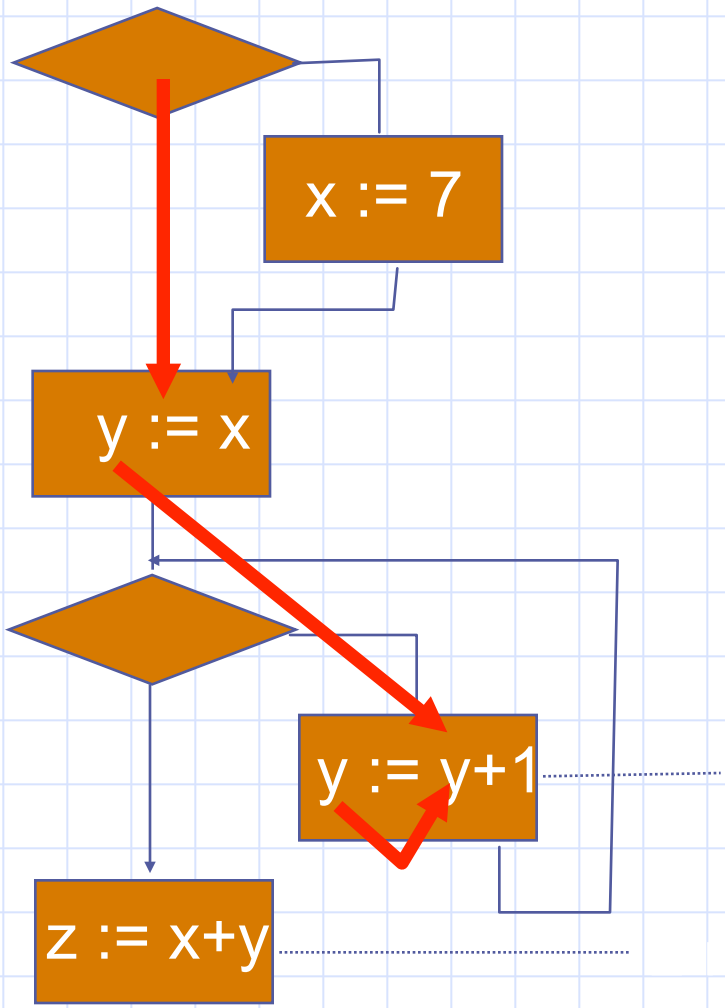
Path coverage

- each execution path considered
  - ac df,
  - abc df,
  - ac dedf,
  - abc dedf, …
- subject to combinatorial explosion

Typical loop coverage criterion would require

- zero iterations (cdf),
- one iteration (cdedf),
- and multiple iterations (cdededed...df)

Data Flow Coverage Criteria



Rationale: An untested def-use association could hide an erroneous computation

Branch vs Condition Coverage

```
if (a && b) {
//c
}
else {
// d
}
```

- Branch coverage
  - test with a = b = true for if branch (c)
  - test with a = false for else branch (d)
- Condition coverage
  - test with a = true, b = false
  - test with a = false, b = true
  - to cover both cases {true,false} for each basic condition {a,b}
- Modified condition/decision coverage (MC/DC)
  - requires that each basic condition be shown to independently affect the outcome of each decision
  - required e.g. by
    - RTCA/DO-178B "Software considerations in airborne Systems …"
    - EUROCAE ED-12B as the European equivalent

Control Flow Coverage Criteria in Practice

- Statement or branch coverage is used in practice
  - Simple lower bounds on adequate testing
- Additional control flow heuristics sometimes used
  - Loops (never, once, many), combinations of conditions
- 100% coverage hard to achieve in practice
- So what do you do if your coverage statistics is too low?
  - Bad idea: quickly create/adapt tests to improve stats
  - Good idea: check what is not covered and why

The Infeasibility Problem

- Adequacy criteria are sometimes impossible to satisfy:
  - Syntactically indicated behaviors (paths, data flows, etc.) are sometimes impossible execute
  - Infeasible control flow, data flow, and data states

- Unsatisfactory approaches:
  - Manual justification for omitting each impossible test case (esp. for more demanding criteria)
  - Adequacy "scores" based on coverage
    - example: 95% statement coverage, 80% def-use coverage

Other Challenges in Structural Coverage

- Inter-procedural and gross-level coverage
  - e.g., inter-procedural data flow, call-graph coverage
- Regression testing
  - maintenance of test suite: adjust vs delete
- Late binding (OO programming languages)
  - coverage of actual and apparent polymorphism
- Fundamental challenge: Infeasible behaviors
  - underlies problems in inter-procedural and polymorphic coverage,
  - obstacle to adoption of more sophisticated coverage criteria and dependence analysis

Summary

Testing is never complete, so when is enough?

- Various coverage criteria
  - Statement / Branch / Path coverage
- Tool support:
  - Eclemma delivers stats + color highlighting
- Code execution does not mean code is correct!
- Code coverage is a “best effort” approach at best,
  - a compromise between the impossible and the inadequate
  - a trade-off between time, effort, effectiveness