SYSC 4101 / 5105

Experimental Software Engineering
Making Enlighted Decisions

Structural Criteria

Analyzing Coverage Data for Decision Making

Experiment A:

- Criteria investigated: All-Edges and All-DU
- Programs: 7 C programs (141-512 LOC)
- Faults: seeding faults leading to 130 faulty program versions
  - created by 10 different people, mostly without knowledge of each other's work; their goal was to be as realistic as possible.
- Test generation procedure
  - was designed to produce a wide range both of test size and test coverage percentages
  - randomly generated test cases.
- They examined the relationship between fault detection and test set coverage / size

Analyzing Coverage Data for Decision Making

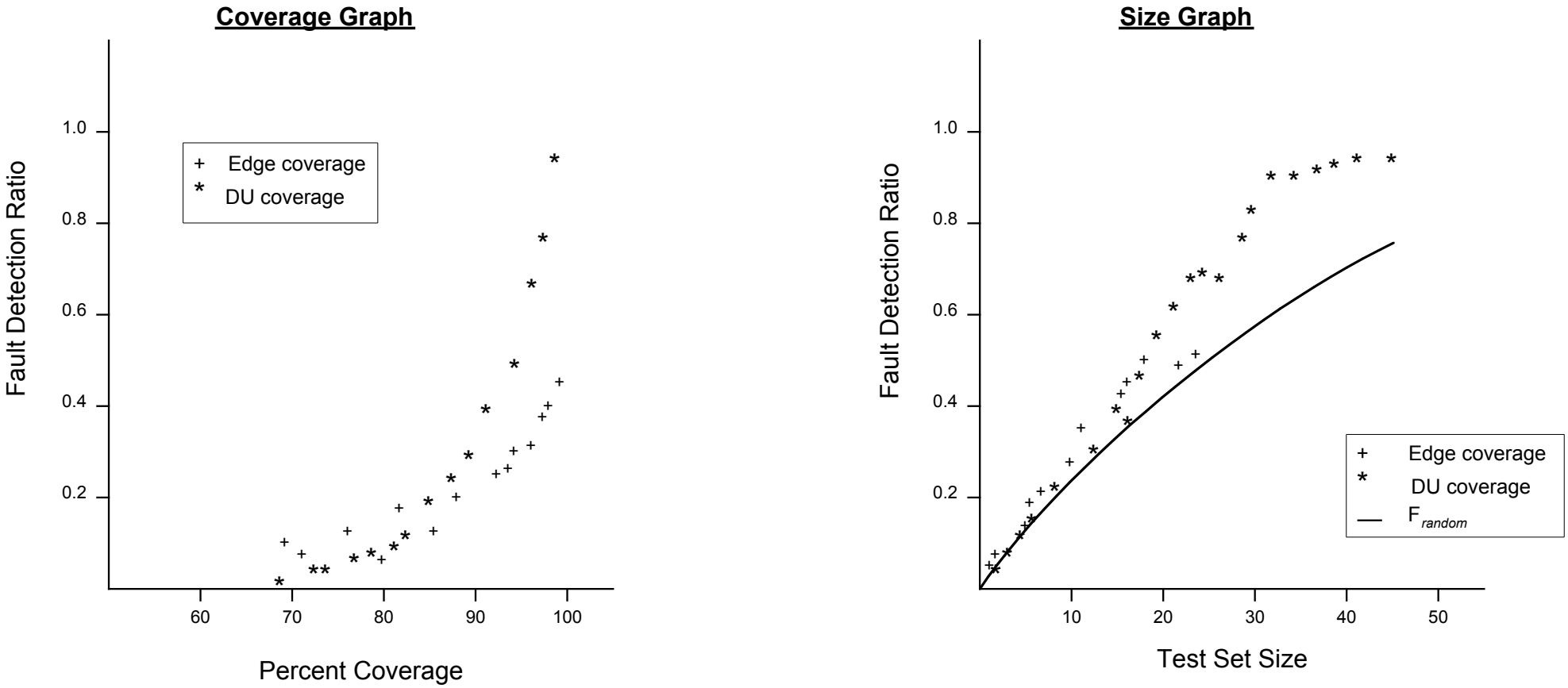


Figure: Fault Detection Ratios for One Faulty Program

Analyzing Coverage Data for Decision Making

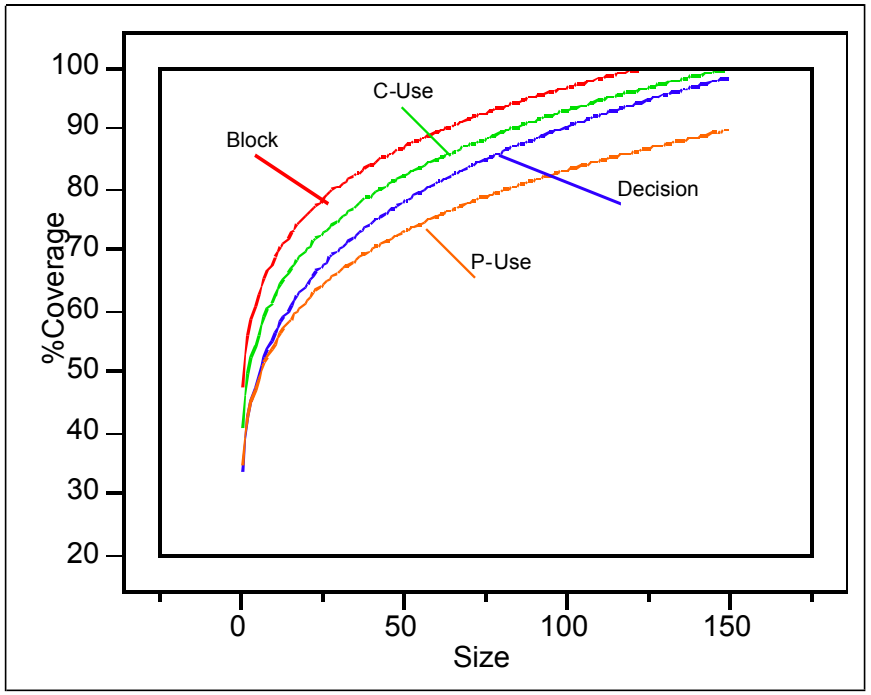
- Both coverage criteria performed better than random test selection – especially DU-coverage
- Significant improvements occurred as coverage increased from 90% to 100%
- 100% coverage alone is not a reliable indicator of the effectiveness of a test set – especially edge coverage
- Test cases based on data flow and control flow criteria are frequently complementary
- As expected, on average, achieving all-DU coverage required significantly larger test sets all-Edge coverage

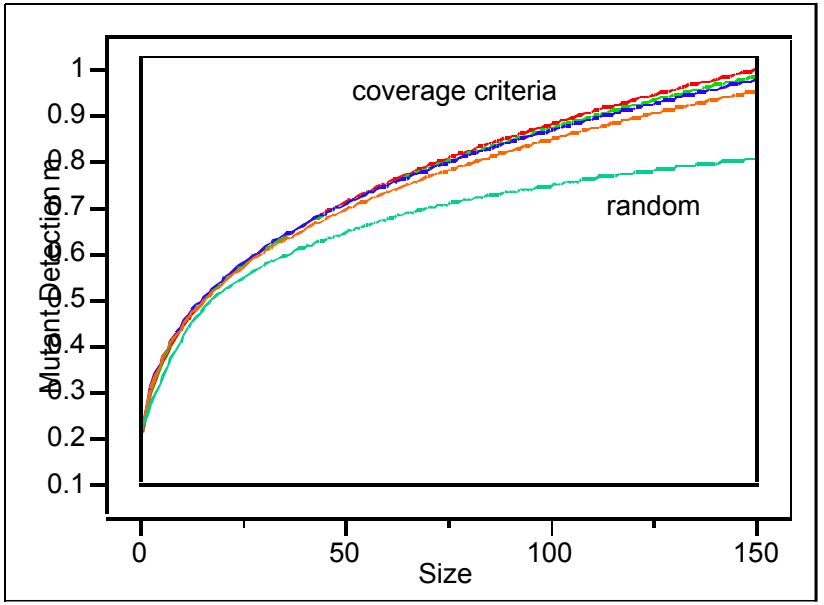
Analyzing Coverage Data for Decision Making

Experiment B:

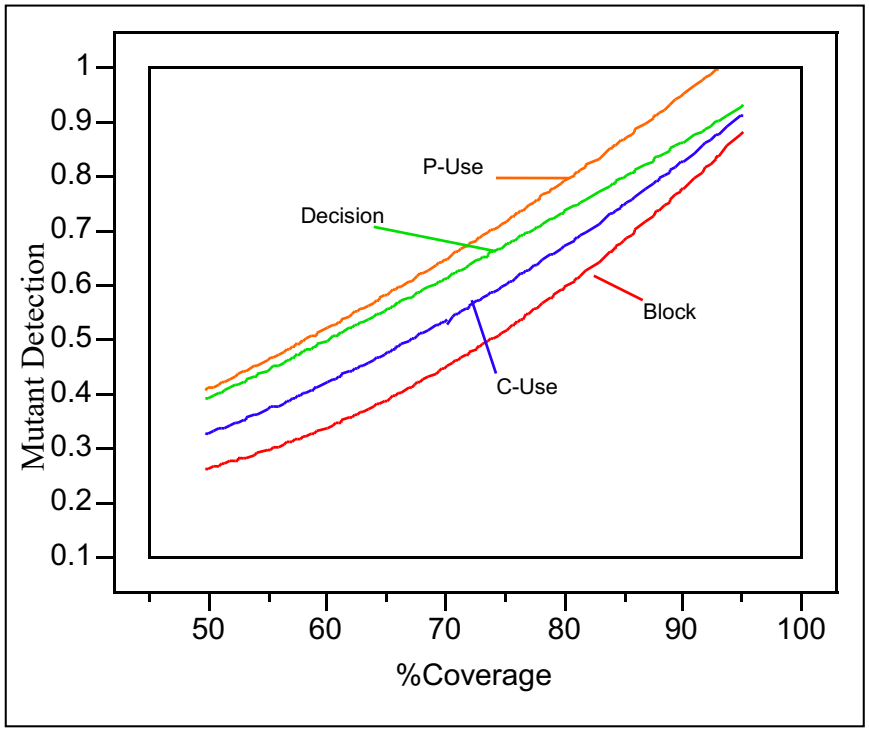
- Criteria investigated: Node (block), Edge, C-Use, P-Use
- Programs: 1 C program (~6,000 LOC)
- Faults: automatically seeded faults (mutants) leading to ~1,200 faulty programs versions
- Test generation procedure
  - ~13,500 test cases
  - Each decision is executed by at least 30 test cases
- They examined the relationship between fault detection and test set coverage / size

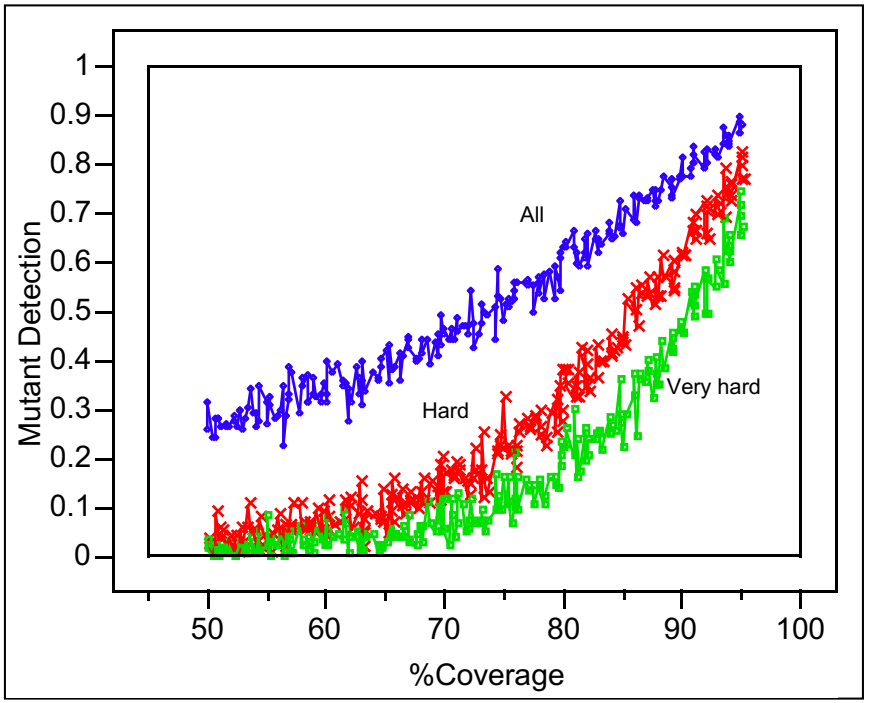
Analyzing Coverage Data for Decision Making





Analyzing Coverage Data for Decision Making





Hard = detected by less than 5% of the test pool
Very Hard = detected by less than 1.5% of the test pool

Analyzing Coverage Data for Decision Making

- No criterion is more cost-effective than the others.
  - However, more demanding criterion (C-Use, P-Use) entail larger test suite (higher cost) and detect more faults.
  - Better cost-effectiveness than random testing
- The probability of detection of faults given a test pool affects the shape of relationships between fault detection, coverage and size
  - A shape as in Experiment A is for "very hard" to detect faults.
  - ➢ The level of coverage to achieve is therefore context dependent.

State-Based Criteria

Empirical Study (1)

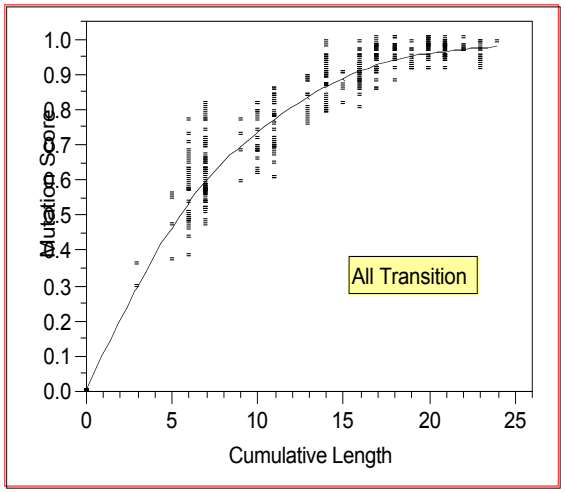
- Small cruise control system (400 C lines, 7 functions, 184 blocks, 174 decisions)
  - Four states: Off, Inactive, Cruise, and Override
- Evaluation Criteria
  - Structural coverage (decision and block coverage)
  - Fault coverage
- Empirical setting:
  - 25 faulty program versions (most of them in the logic that implemented the state machine)
  - Tests were created independently from the faults, by different people (manually)
  - As a comparison, 54 test cases were generated randomly
- Results
  - Transition coverage is more cost-effective than random (same effectiveness but less expensive)

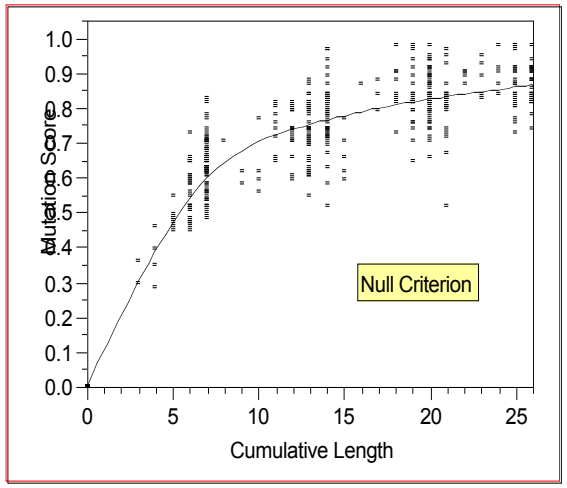
	Random	Transition
# TC	54	12
Faults found	15	15
Faults missed	9	9
Percent (mutation score)	62.5%	62.5%

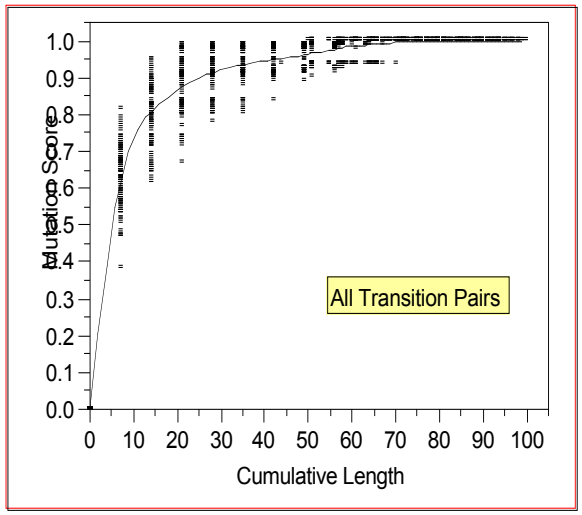
Empirical Study (2)

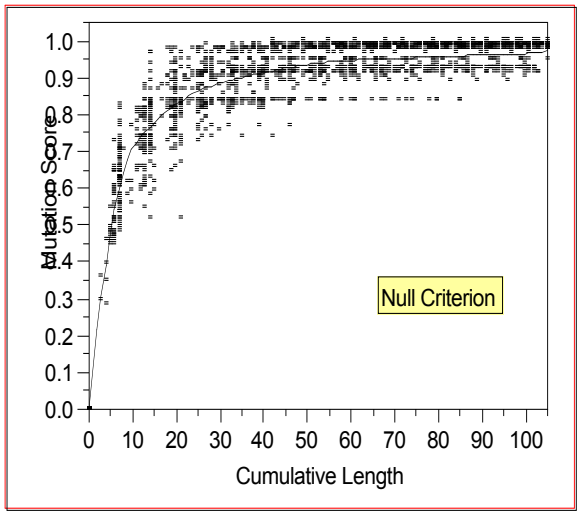
- Two software components
  - Cruise Control Software (only event based)
  - OrdSet Class (events + input parameter values)
- Evaluation on faulty versions (roughly 100)
- Criteria
  - Transition
  - Transition pairs
  - Transition tree
  - Full predicate (will be discussed later, in another part of the course)
- 100 adequate test suite for each criterion (except transition tree)

Cruise Control—Transition (pairs)







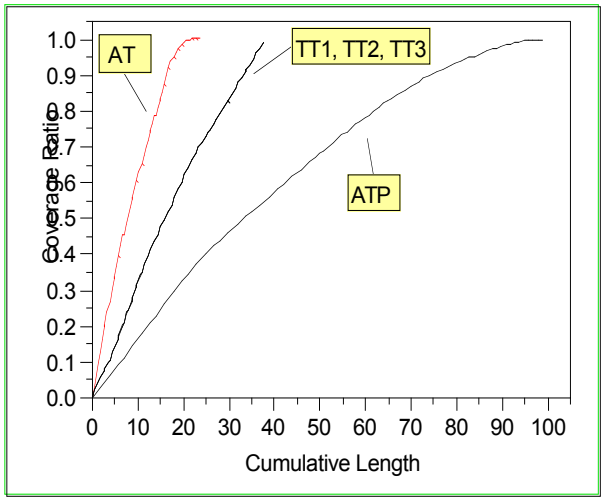


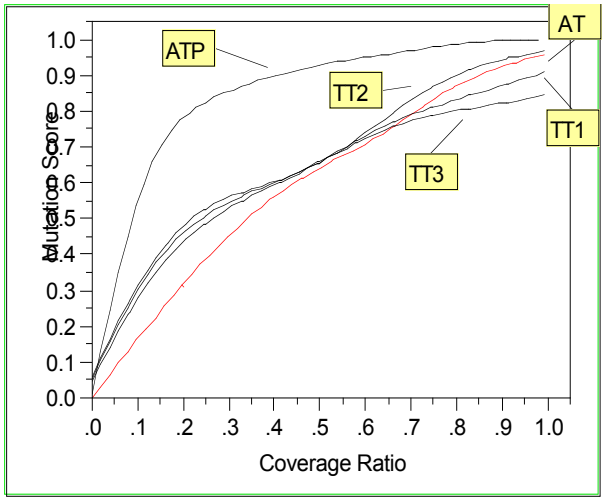
- Graphs show the mutation score as we execute more and more test cases until we reach full (100%) coverage

Observations

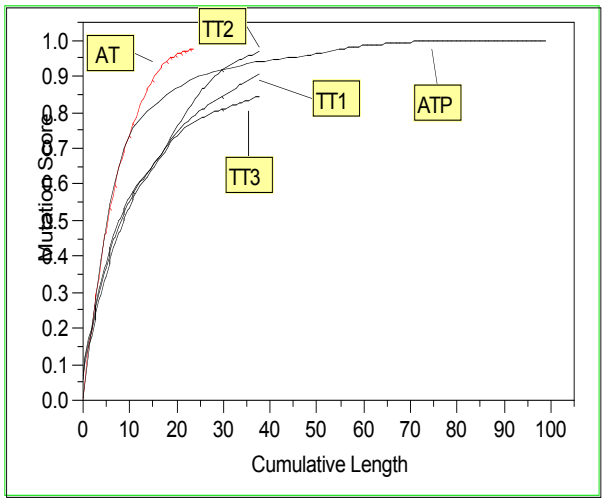
- Criteria do better than random
- Transition Pairs coverage much more expensive than Transition coverage (100 vs. 25)
- But Transition Pairs coverage more effective than Transition coverage (100% vs 95%)

Cruise Control—Comparing Criteria





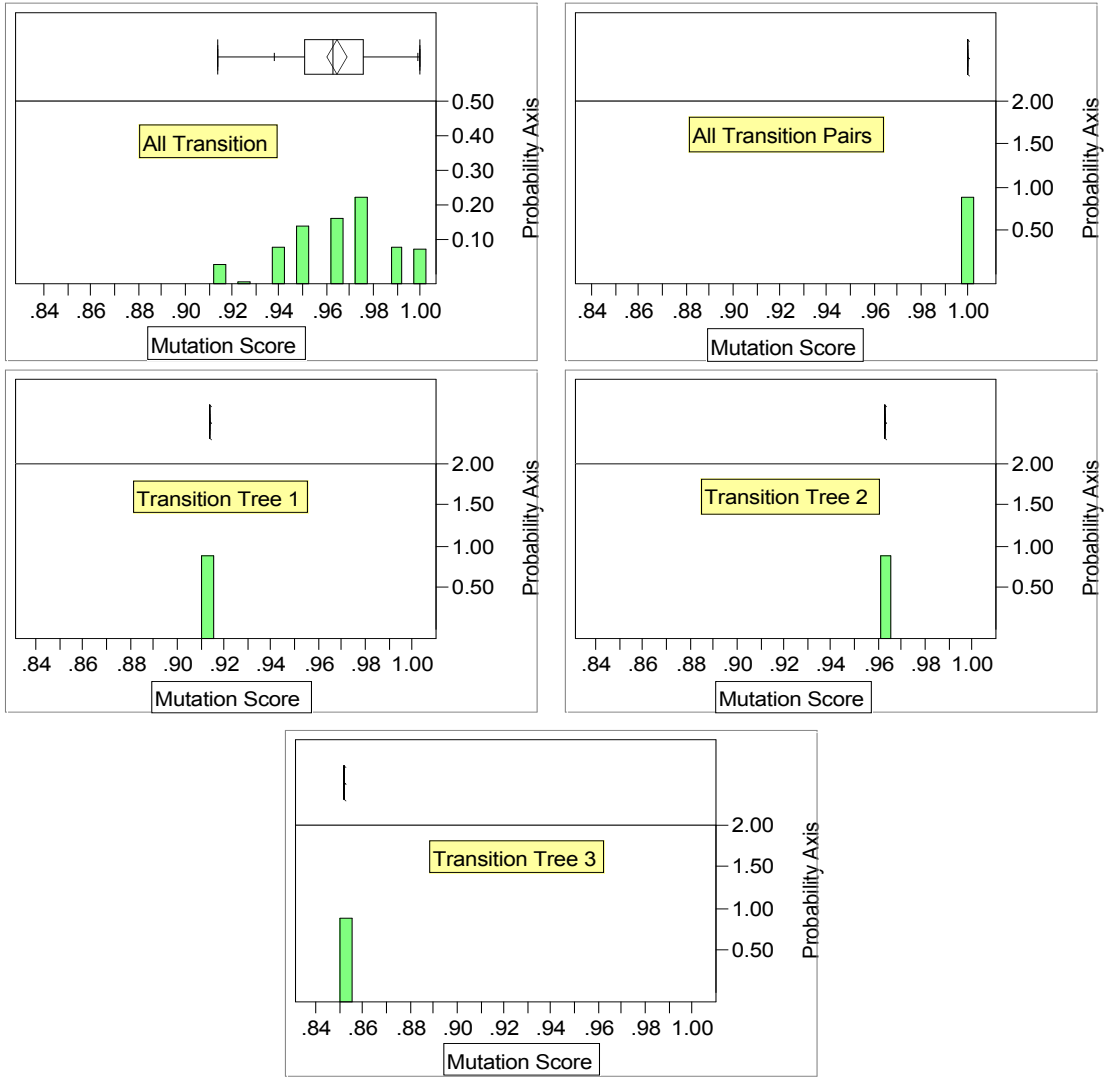
AT=Transition coverage
ATP=Transition Pair
TTi=Transition Tree



Observations:

- Different transition trees (3) with same cost but very different effectiveness (85% to 96%, roughly)
  - They trigger the cruise control on a stopped car!
- But good compromise between Transition and Transition Pair coverage criteria

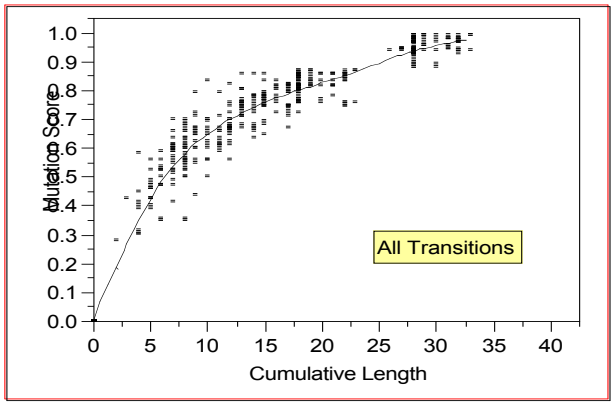
Cruise Control—Adequate Test Suites only

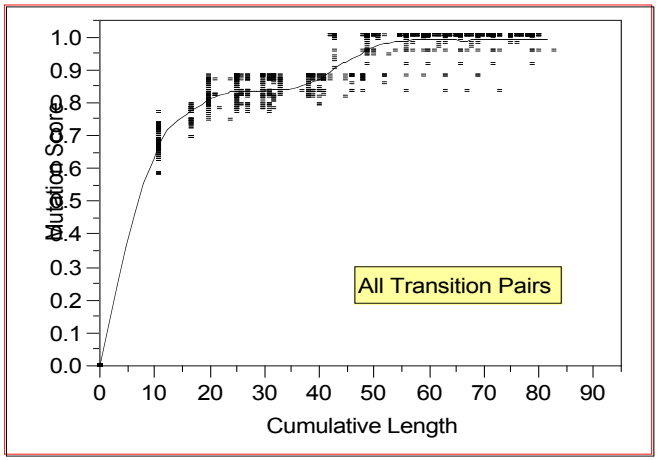


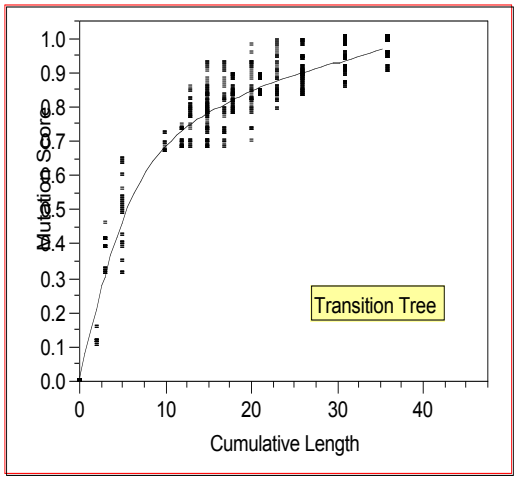
Observations:

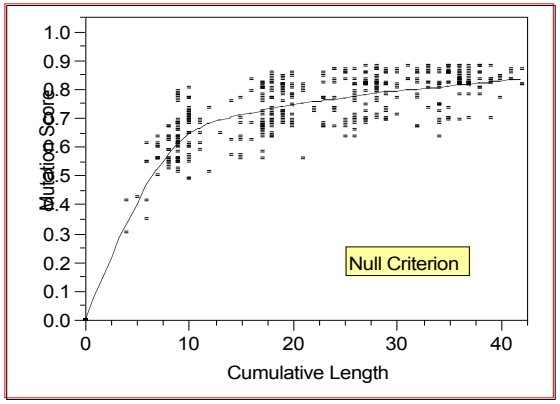
- Transition coverage has a huge variability (one can really be unlucky)
- No room for variability with the Transition pairs
- Variability with Transition Tree (already discussed)

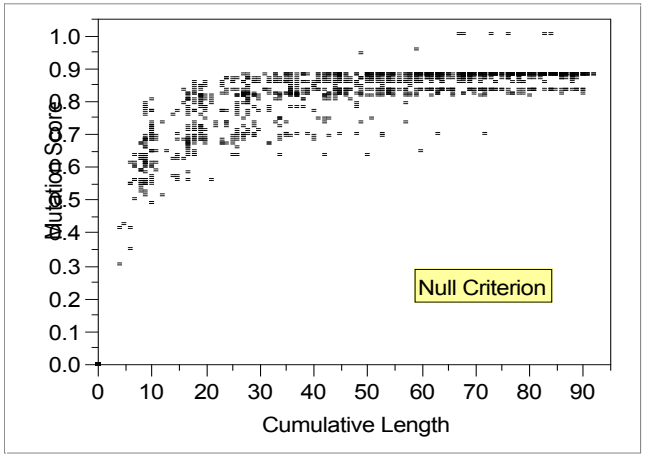
OrdSet—Criteria

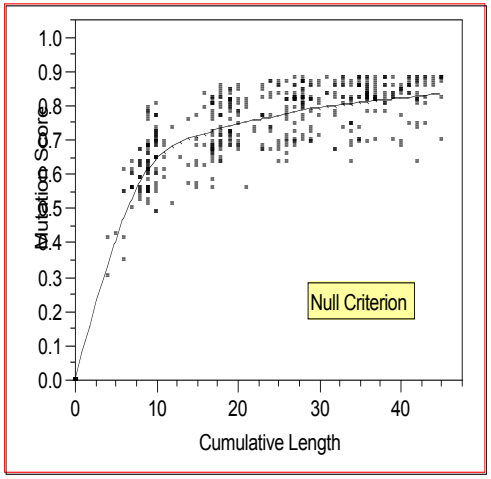






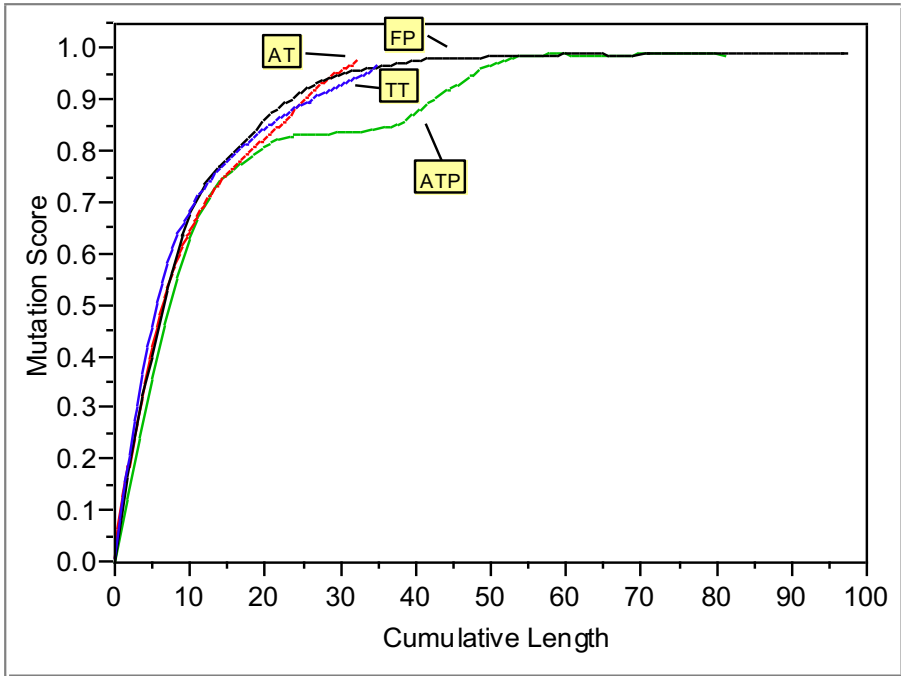


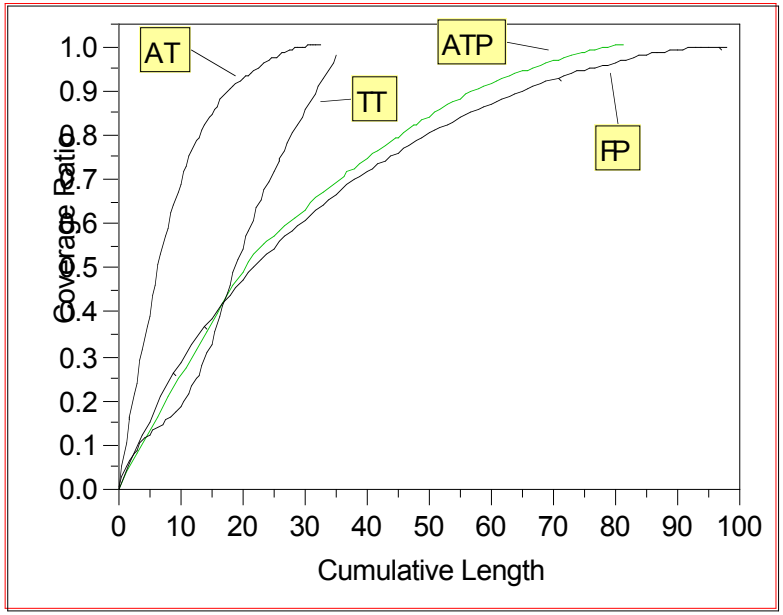


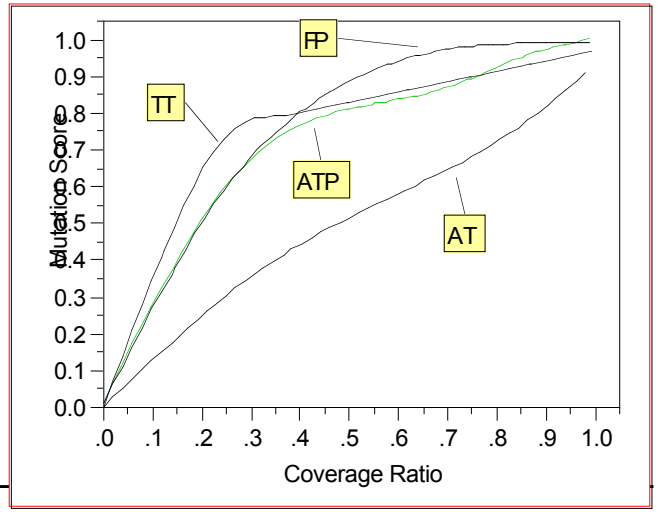


Same observations as before (variation for transition tree due to input parameter values)

OrdSet—Comparing Criteria



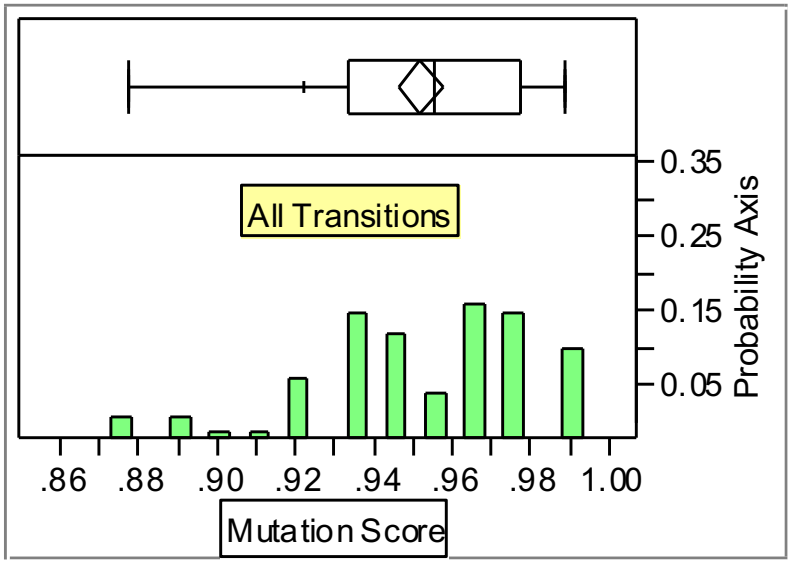




Same observations as before:
Transition Tree is a good compromise between very expensive but very effective criteria (e.g., Transition Pair—TP) and very cheap but not effective criterion (Transition—AT).

(FP=full predicate, not discussed here)

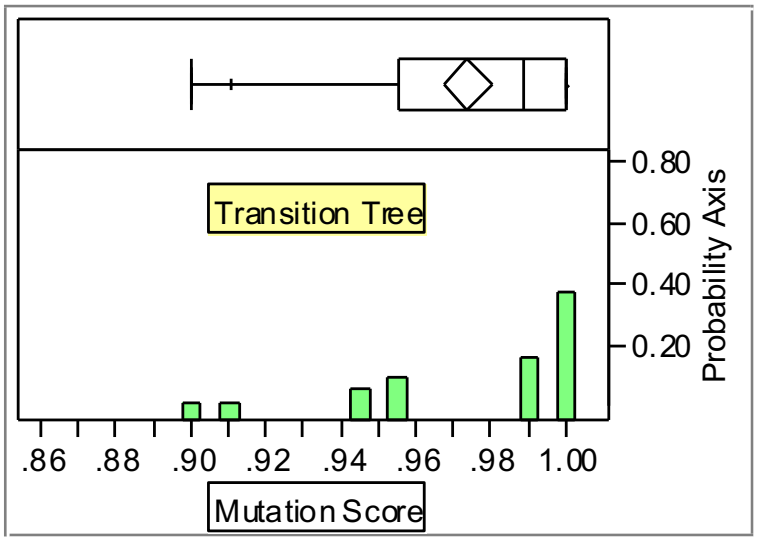
OrdSet——Adequate Test Suites only

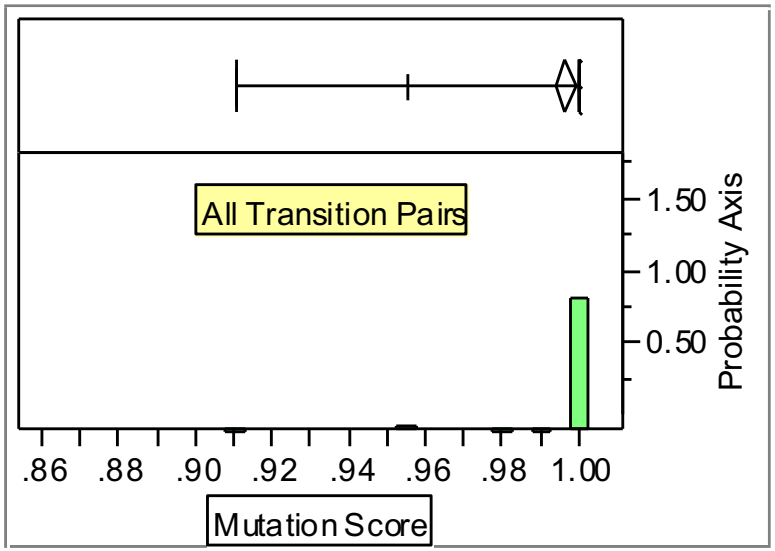


Observations:

- Certainty with Transition Pair
- A lot of uncertainty with Transition
- Medium uncertainty with Transition Tree

Confirming that Transition Tree is perhaps a good compromise





Summary of Results

- Transition Coverage probably not reliable enough as an indicator of fault detection
- Transition Pair Coverage highly reliable but also substantially more expensive
- Transition Tree gets mixed results, depending on the statechart properties: guard conditions, etc.
  - Good compromise when many guard conditions.
  - Slightly more expensive but significantly more effective than Transition.
  - Where alternative transition trees are possible, one should be careful to select the one that exercises the code in the most realistic and complete manner

Logic Criteria

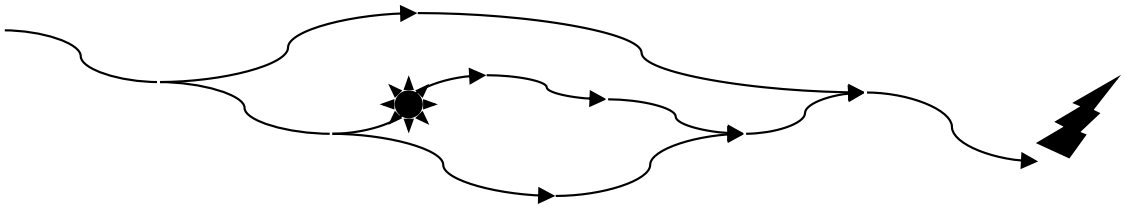
Empirical Study

- TCASII, aircraft collision avoidance system
  - 20 boolean expressions formed the specifications (in modified statechart notation)
  - On average 10 distinct literals per expression
- Faults seeded in the specification
- Study of CUTPNFP
- Random selection of test cases (same size as CUTPNFP) leads to an average mutation score of 42.7%
- CUTPNFP is therefore doing much better with an average of 97.9%.

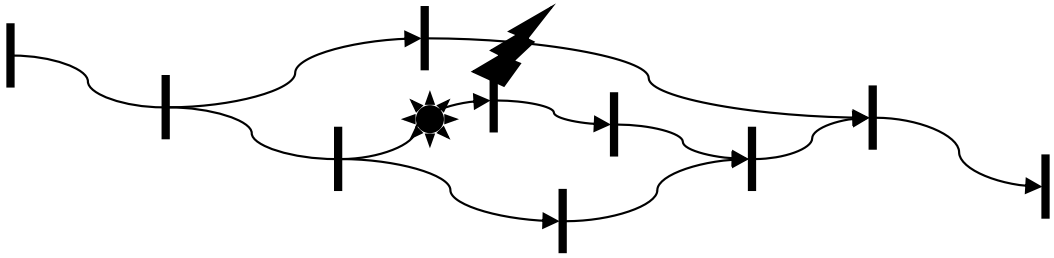
Contracts and Oracles

Are Invariants/Contracts Good Oracles?

- Diagnosis of failure is time-consuming in OO software because of the complexity of methods interactions
  - i.e., lots of methods are involved in any simple execution



- How can analysis contracts (invariant, pre and post-conditions) be used to help the isolation of faults?



Are Invariants/Contracts Good Oracles?

- Invariants/Contracts
  - OCL in the context of UML, Java assert mechanism, The Java Modeling Language (JML), …
- Different levels of details
  - Highest (as defined during Analysis)

```
if A1 then B1
else if A2 then B2
else B3
```

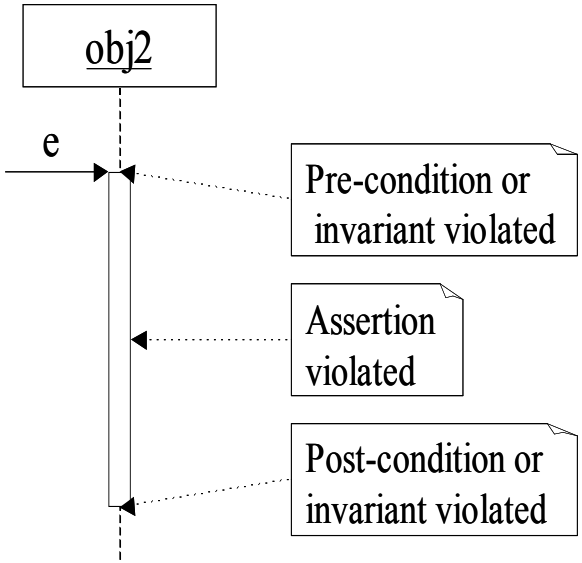
  - Intermediate (check the condition for the standard situation, say `A1`)

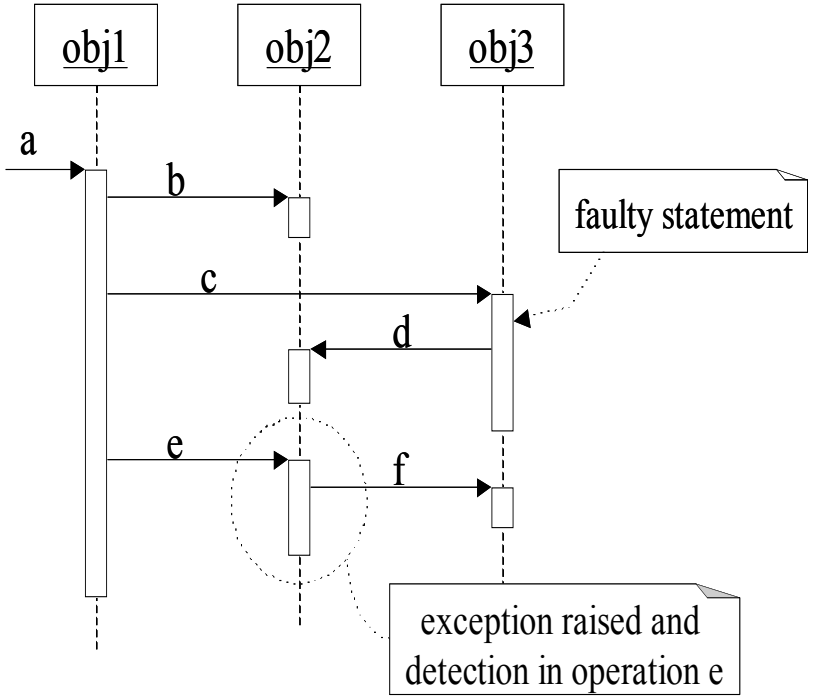
```
if A1 then B1
else B2 or B3
```

  - Lowest (ignore the conditions, check only the possible results)

```
B1 or B2 or B3
```

Diagnosability measure





Exception raised	Methods analyzed	Measure
at `e`'s entry	a, b, c	3
during `e`, before the call to `f`	e, a, b, c	4
during `e`, after the call to `f`	e, f, a, b, c	5

Experimental Setting

- Case study: an Automated Teller Machine
  - 21 classes, 2200 LOCs
  - with Analysis contracts (defined using OCL)
- Seeding of faults: 112 faults (OO and non-OO)
  - Number deemed sufficient considering the case study
    - In terms of mutation operators used (17), and classes seeded (11)
- Test cases: Input Domain Testing
- Execution:
  - 4 different ATM case studies: without contracts (only oracle), with contracts at three different levels of details.
  - Diagnosability measure for each mutant in each execution.

Results

- 112 mutants
  - 99 mutants killed by oracles
  - 84 mutants killed by contracts (5 equivalent mutants killed)

		# of mutants considered	Average diagnosability
Oracle only		79	13.3
Contracts	Highest	79	1.35
	Intermediate	75	1.36
	Lowest	74	1.37

- Using contracts (at any level of detail) improves a lot diagnosability (one order of magnitude)
  - Significant effort savings during debugging
- Using simple contracts is sufficient