Programs, Test Data, and Oracles: Revisiting the Foundations of Software Testing

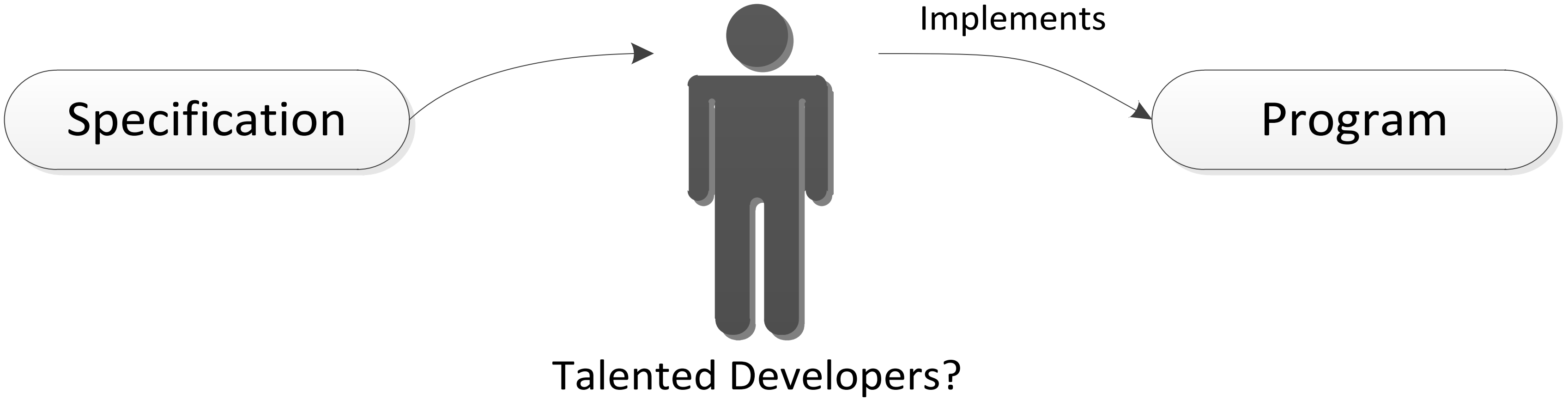
Mats P. E. Heimdahl

University of Minnesota Software Engineering Center
Department of Computer Science and Engineering
University of Minnesota
4-192 EE/CS; 200 Union Street SE
Minneapolis, MN 55455

UNIVERSITY OF MINNESOTA

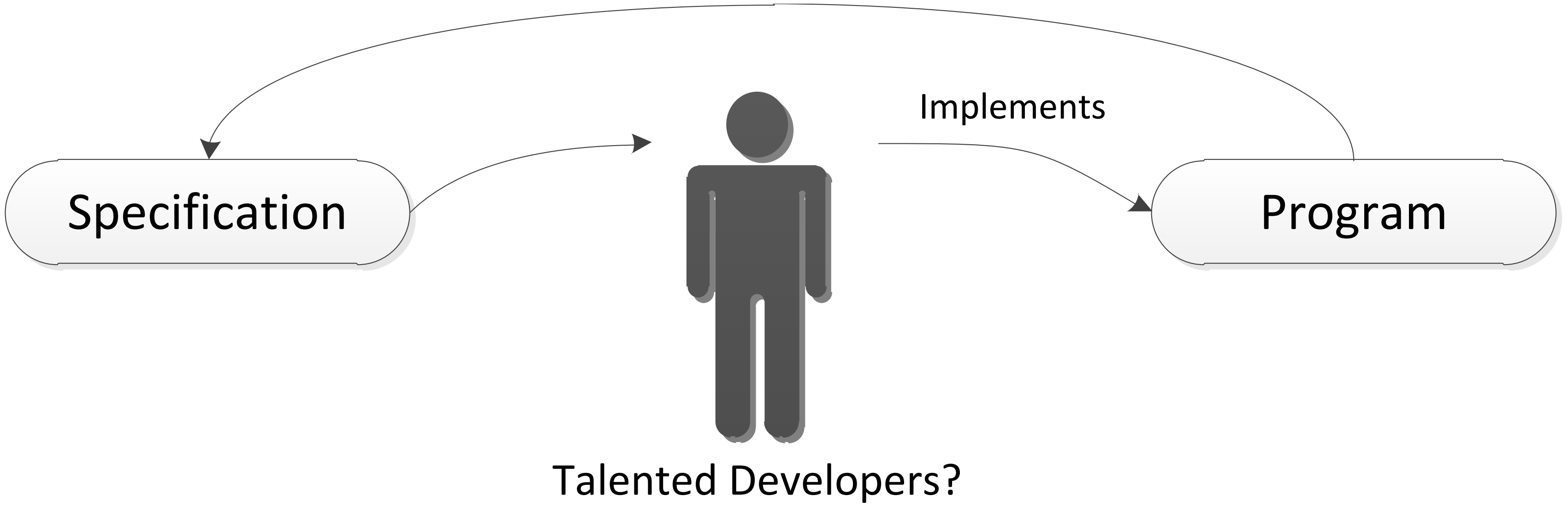
Software Engineering Center

Software Development

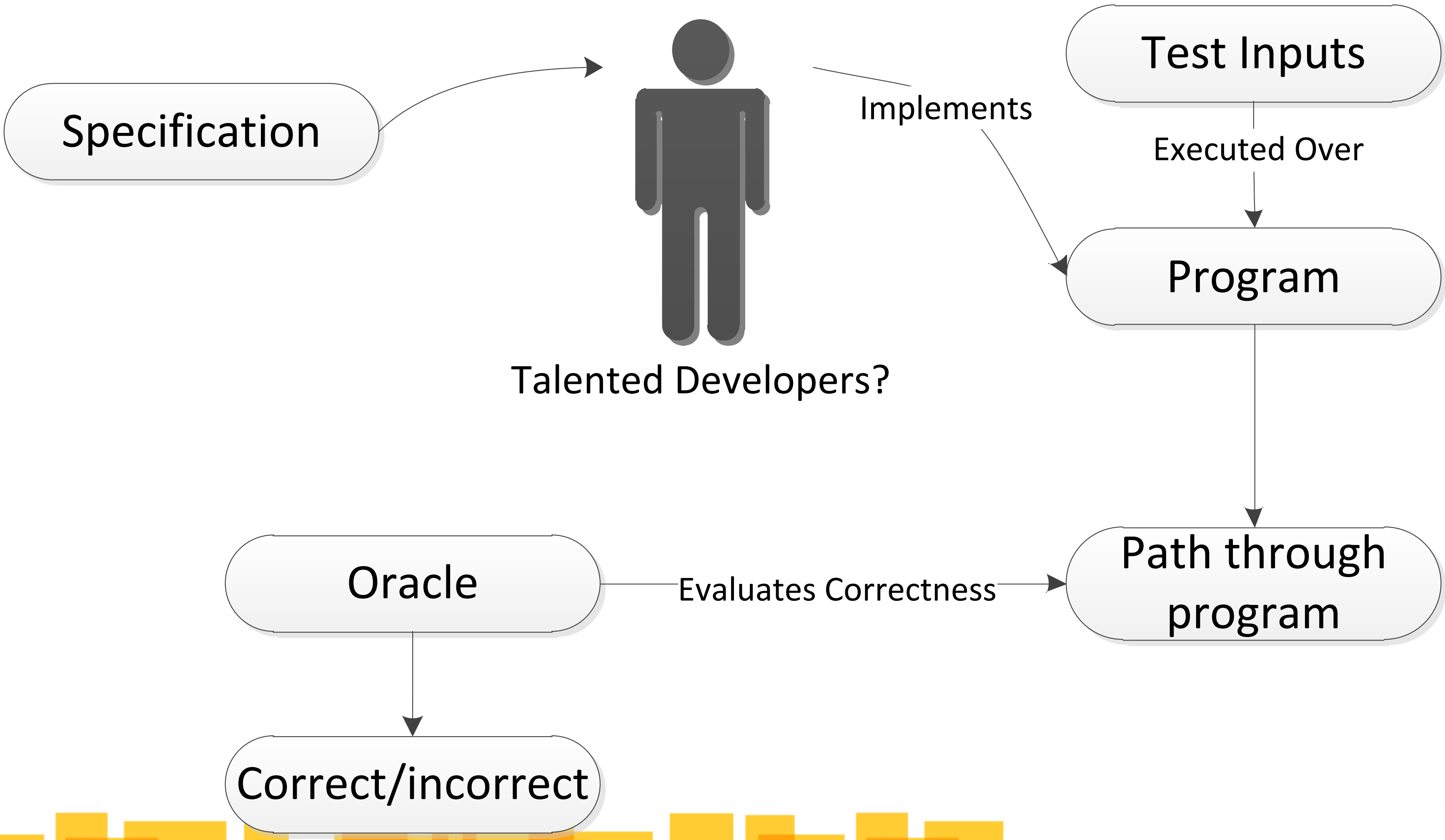


The Big Question

Does the program accurately represent the specification?

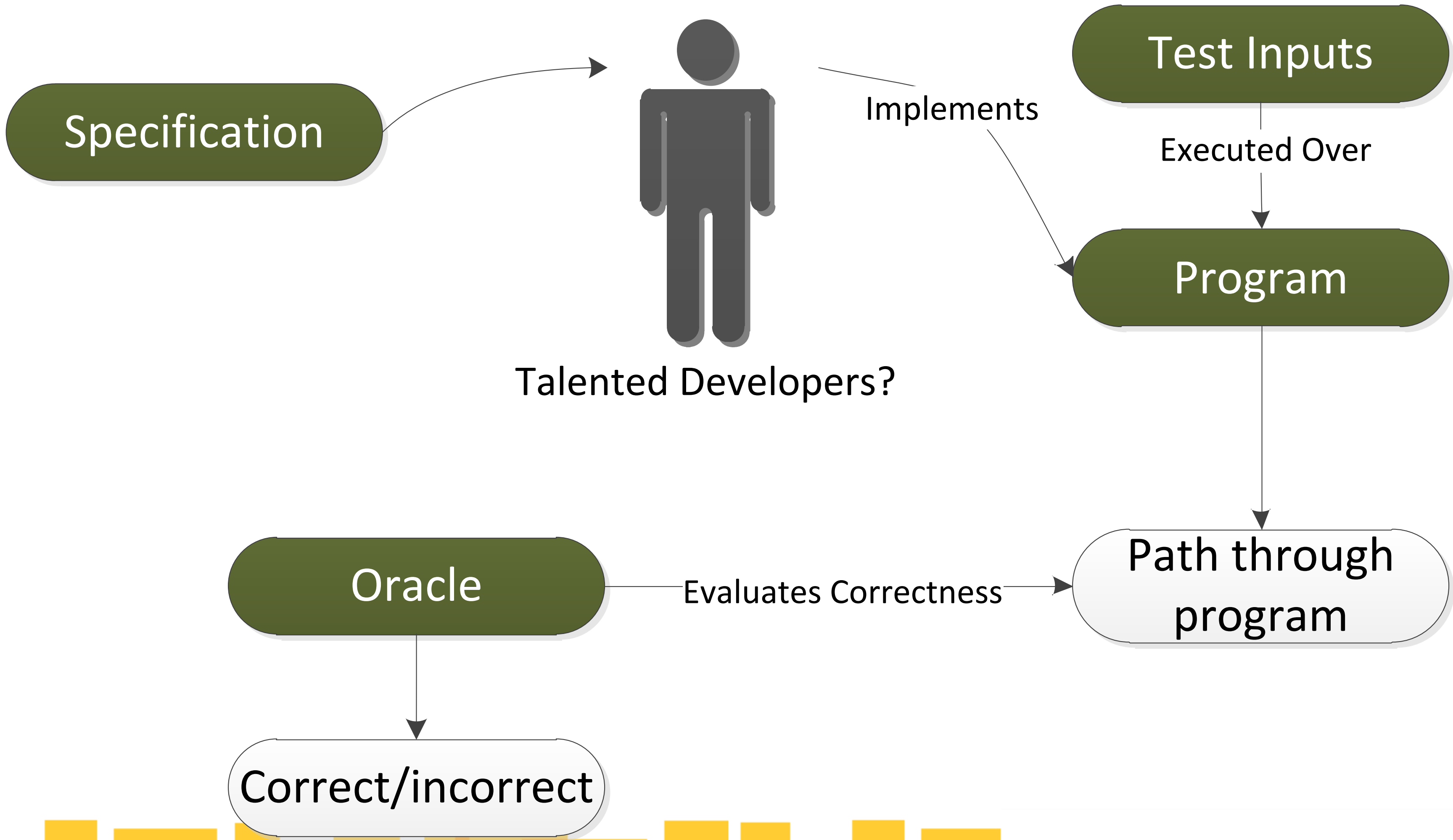


Testing Process



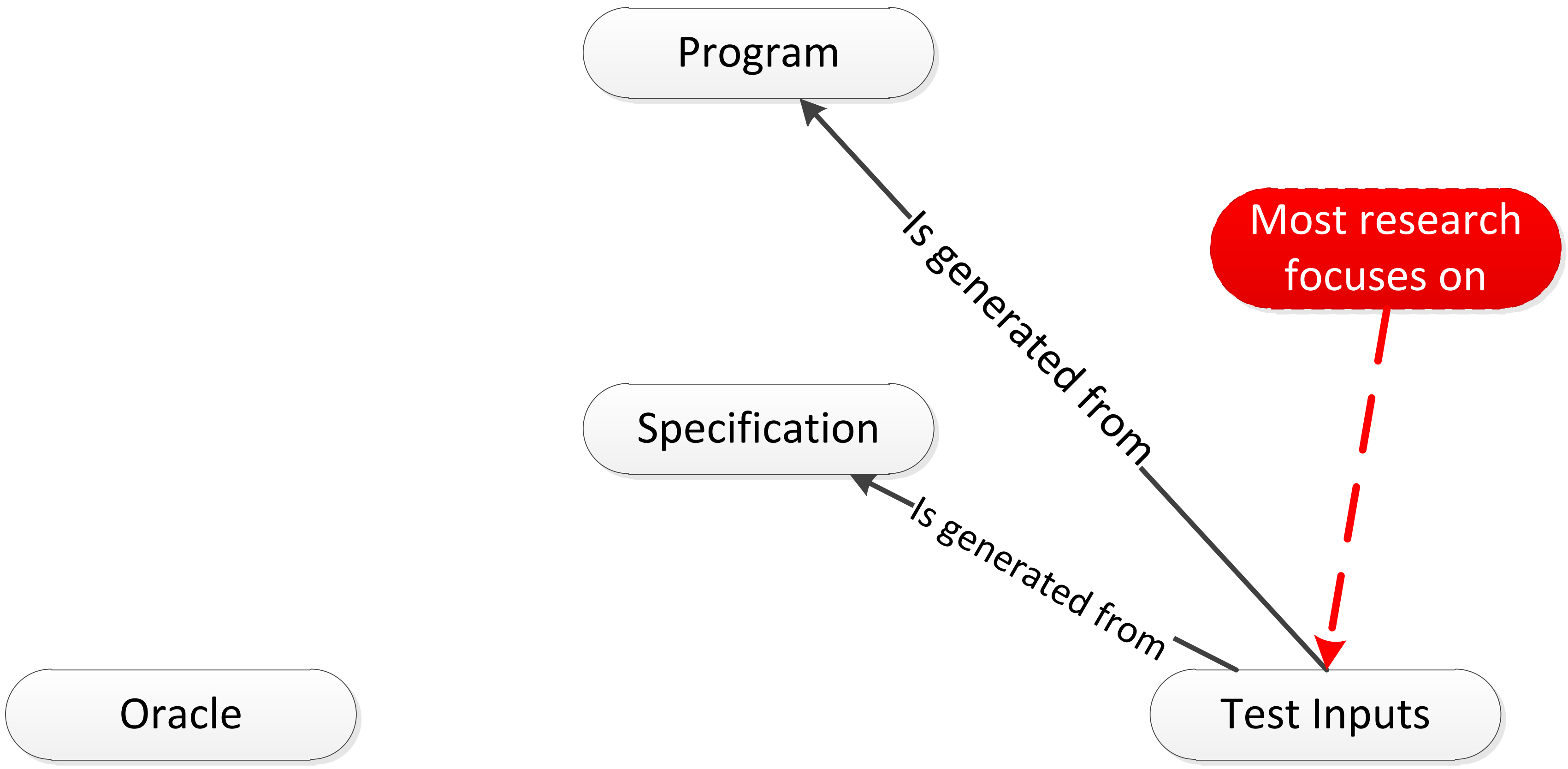
UNIVERSITY OF MINNESOTA
Software Engineering Center

Testing Process



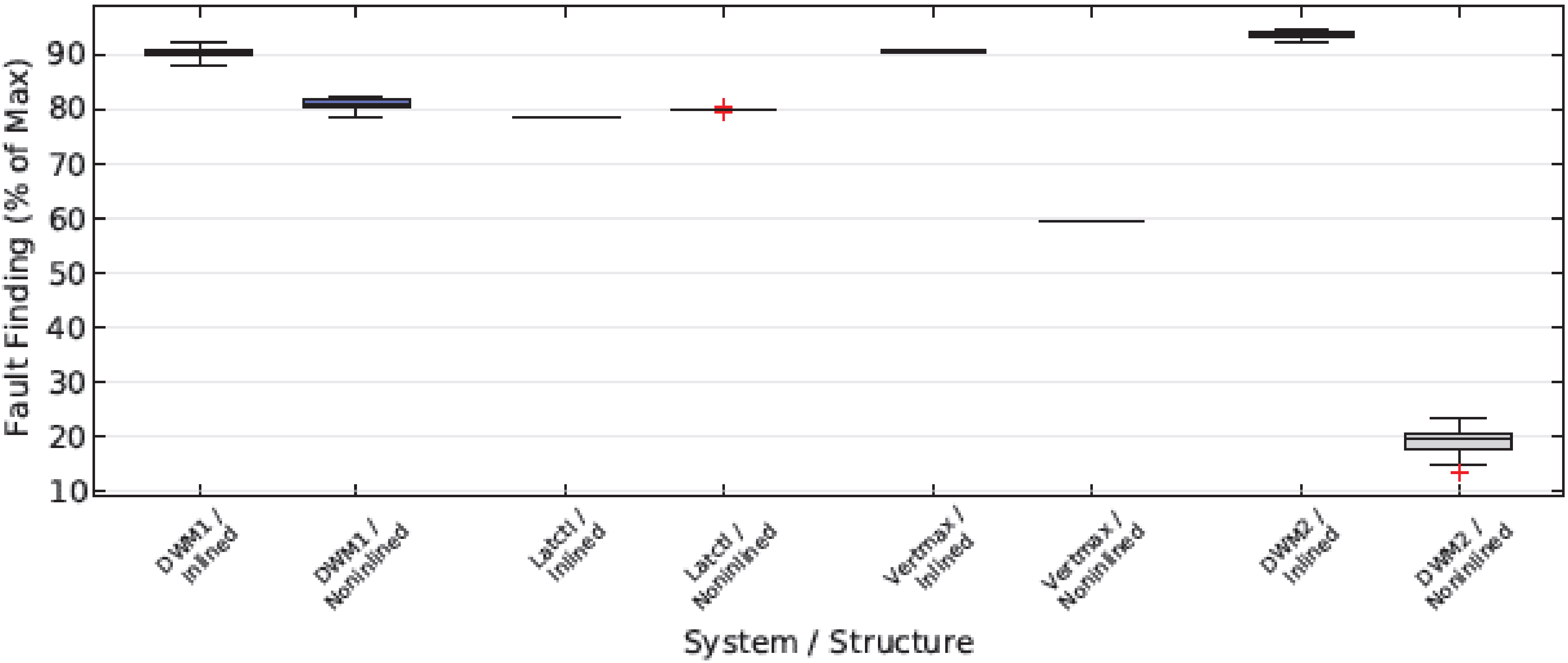
Domains of Concern

Testing Artifacts – In Practice

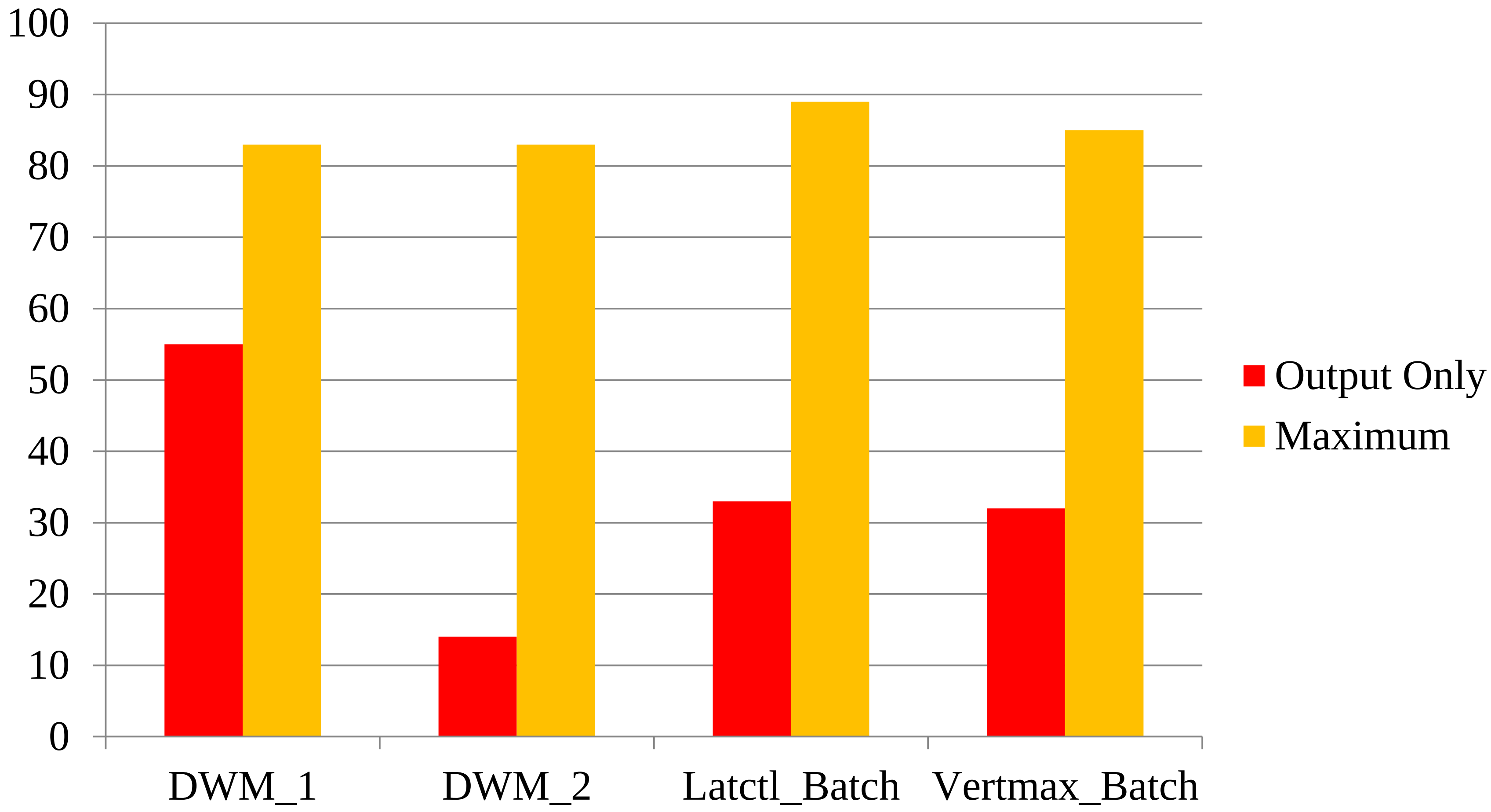


Fault Finding; MC/DC

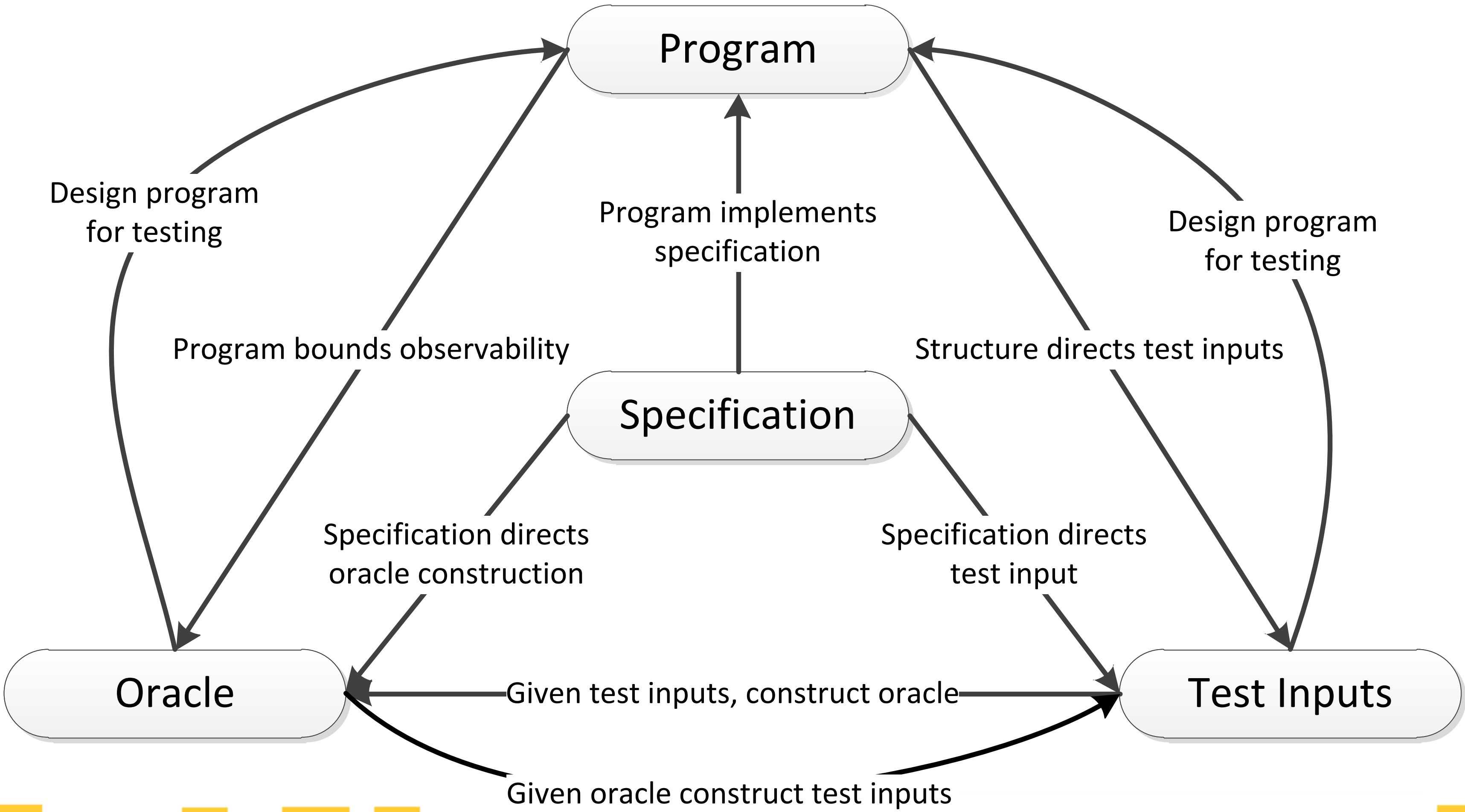
- Program structure matters



Fault Finding; Branch Coverage

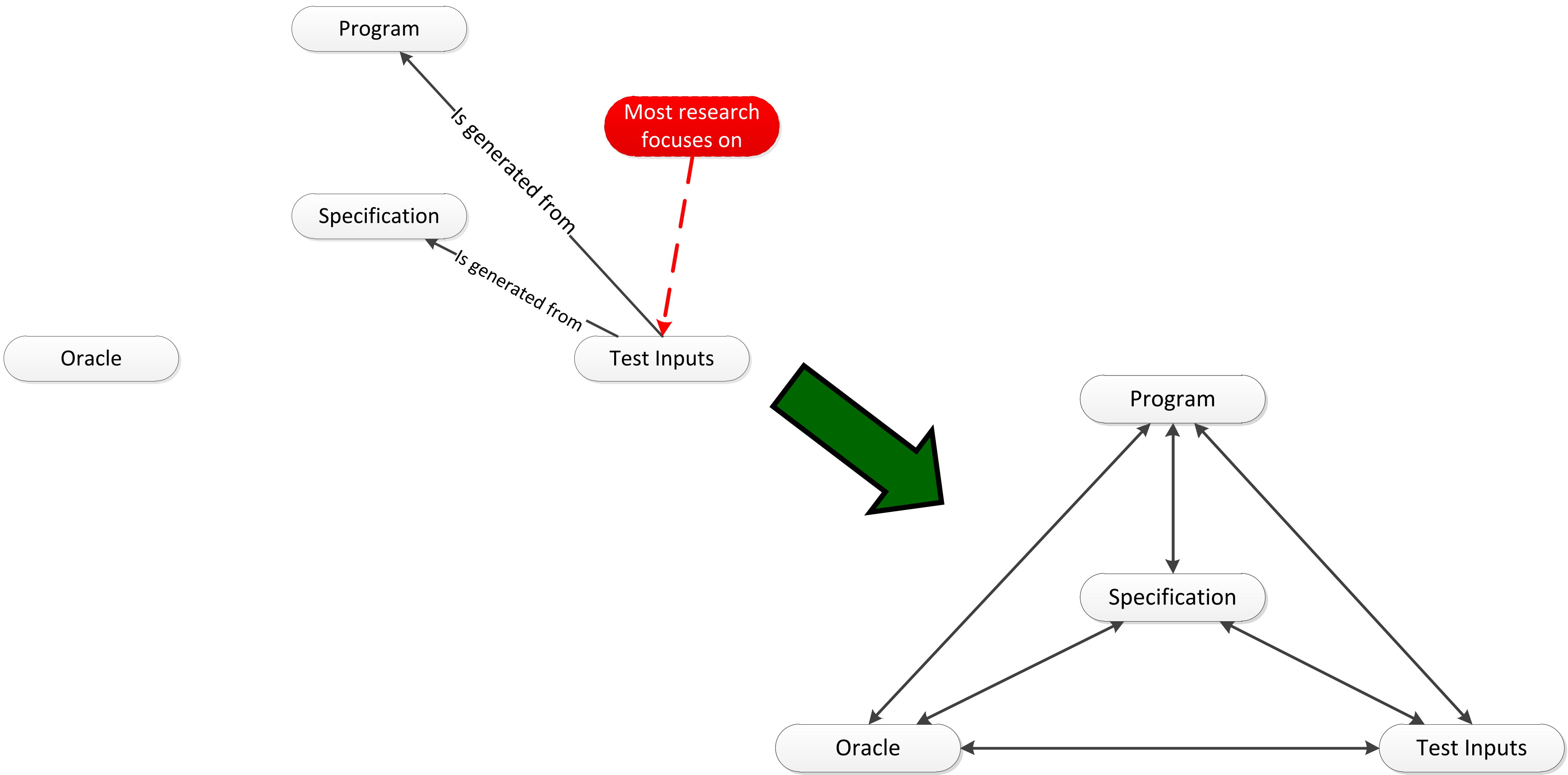


Testing Artifacts - Relationships



University of Minnesota
Software Engineering Center

Testing Artifacts – Broaden View



Importance of Understanding Relationship Between Artifacts

Unexplored testing artifacts represent potential for improving testing effectiveness

Uncontrolled factors represent a threat to validity of empirical studies

Poorly understood factors may result in misapplication of methods

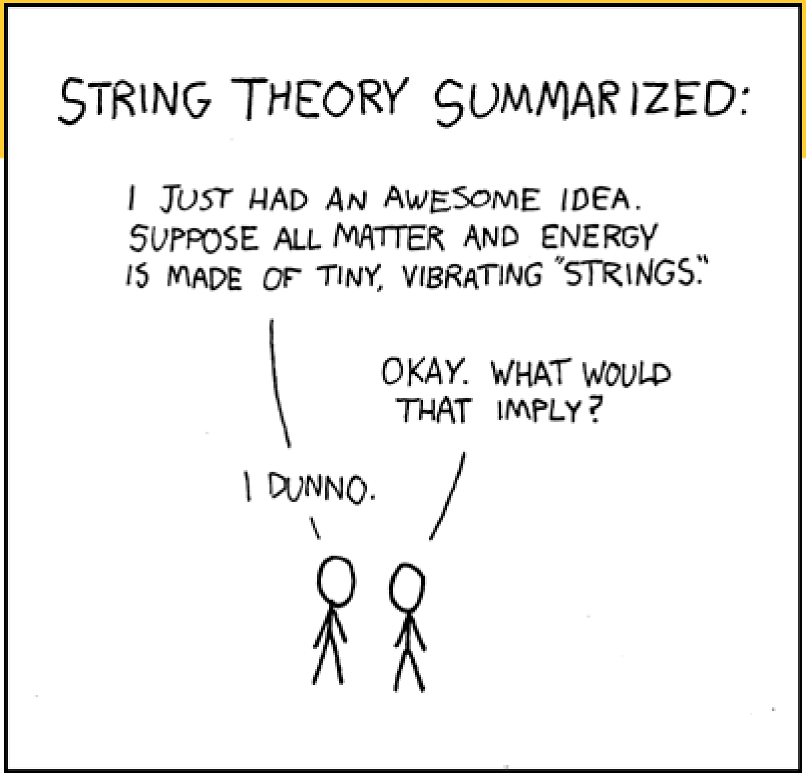
Acknowledgements

- I have not done this alone
  - Matt Staats, Google, Zurich
  - Mike Whalen, U of Minnesota
  - Ajitha Rajan, Edinburgh
  - Gregory Gay, U of South Carolina
  - Rockwell Collins Inc.
    - Steve Miller, Darren Cofer

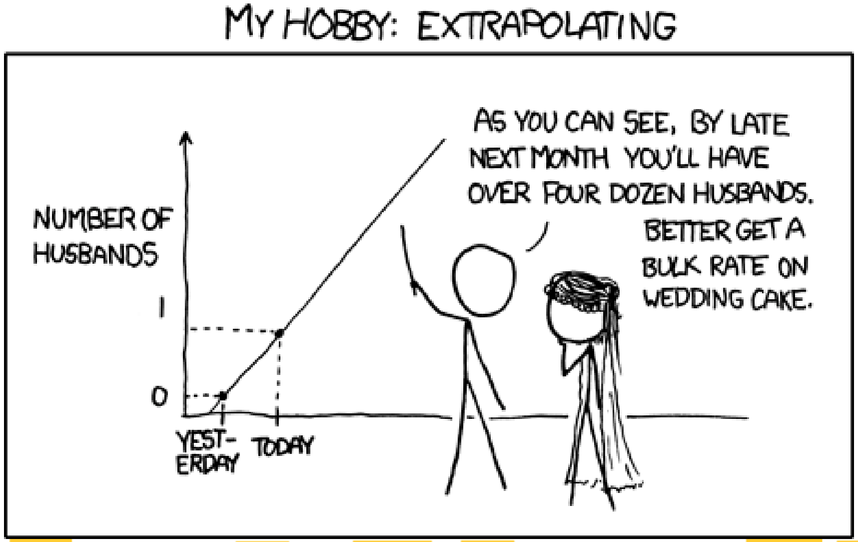
Funded by CNS-0931931 and CNS-1035715

Two Approaches

Theory of Testing



Empirical Studies



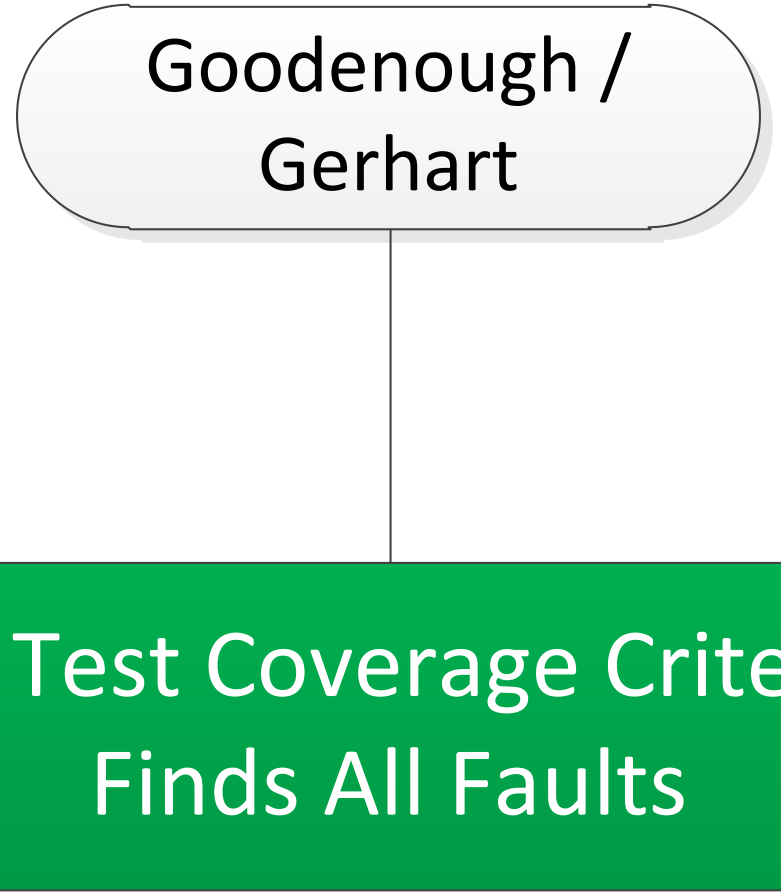
Adopted from Matt Staats

UNIVERSITY OF MINNESOTA
Software Engineering Center

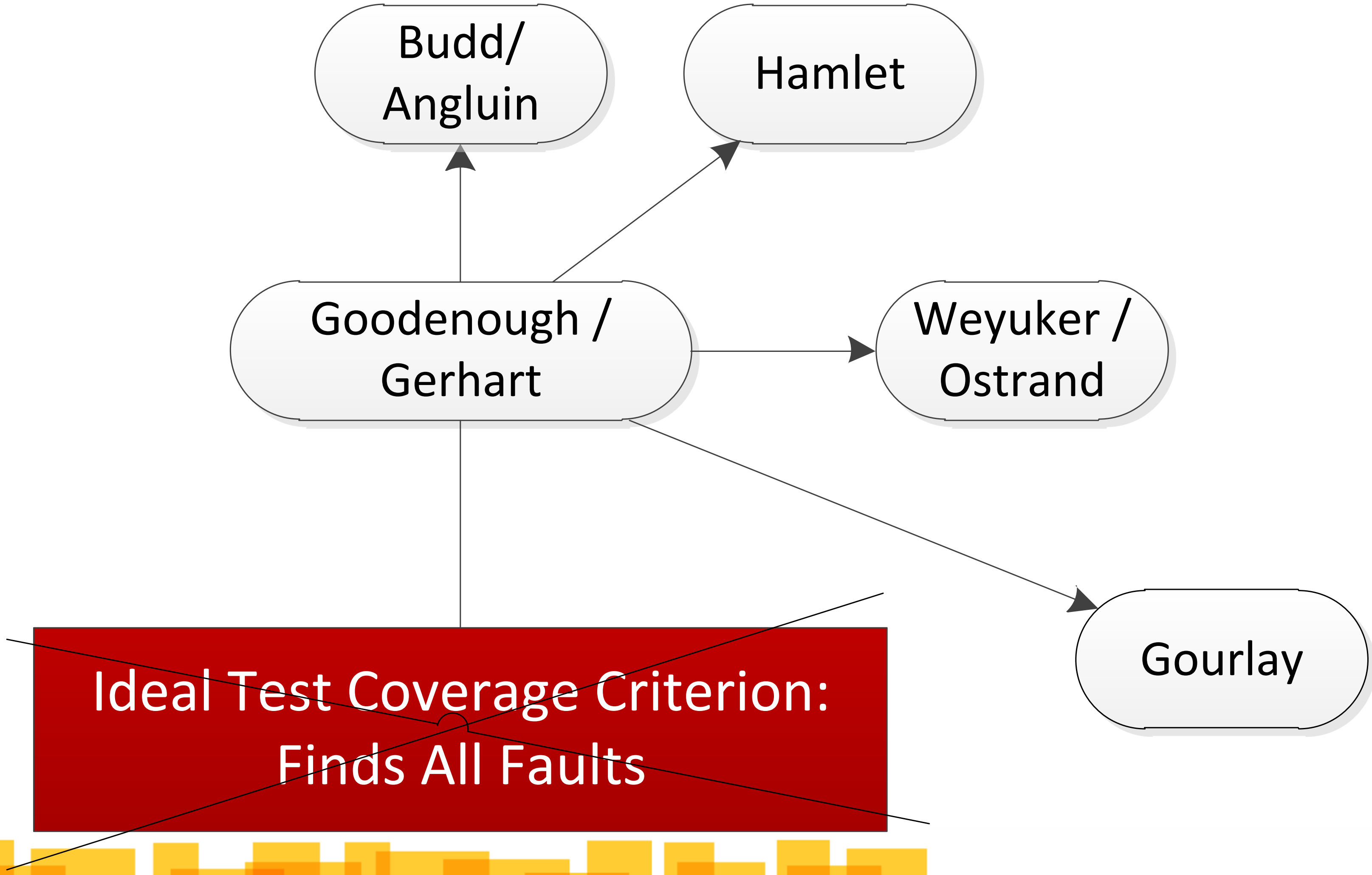
Two Approaches

Theory of Testing

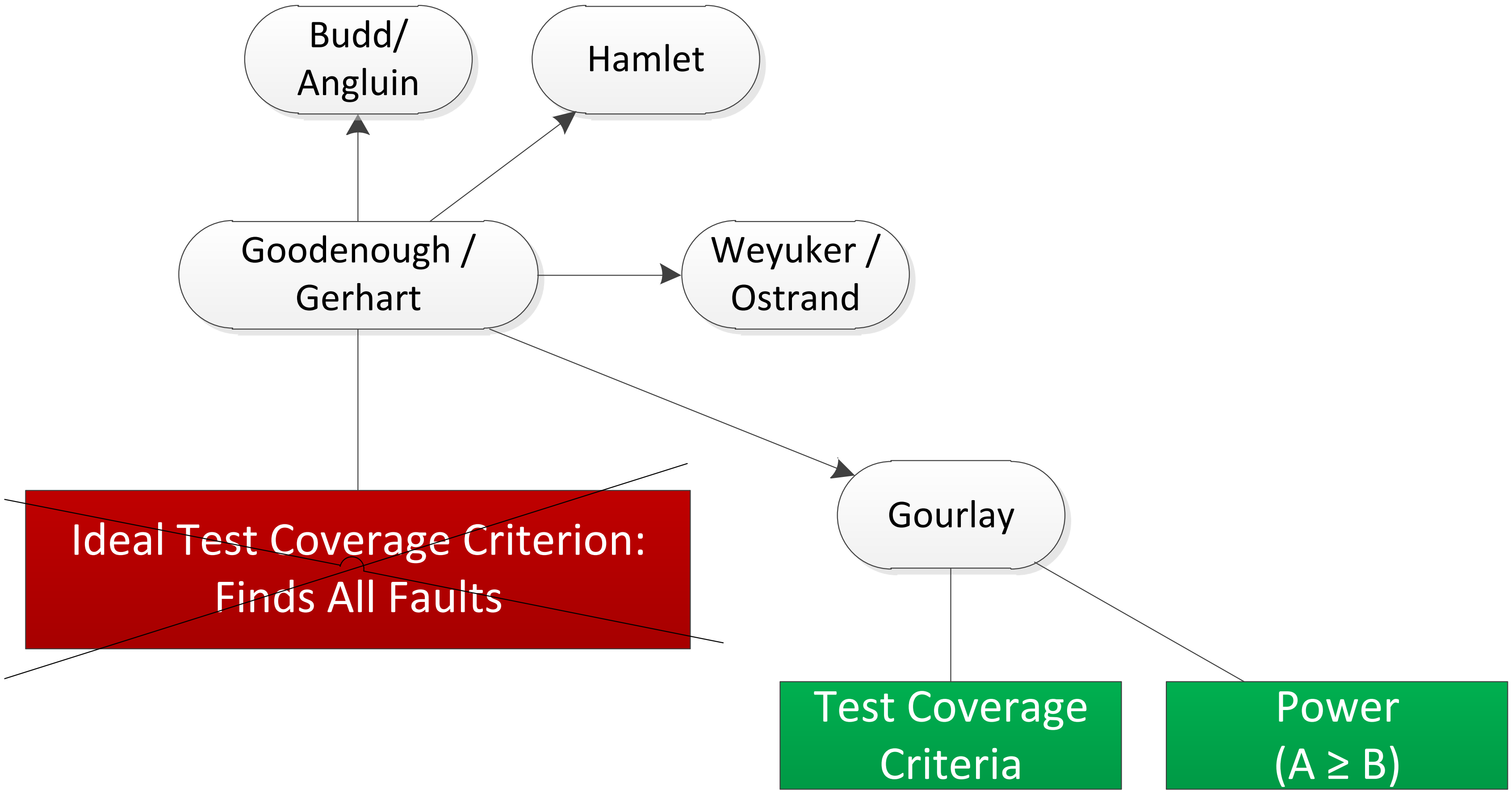
Theory of Testing - History



Theory of Testing - History

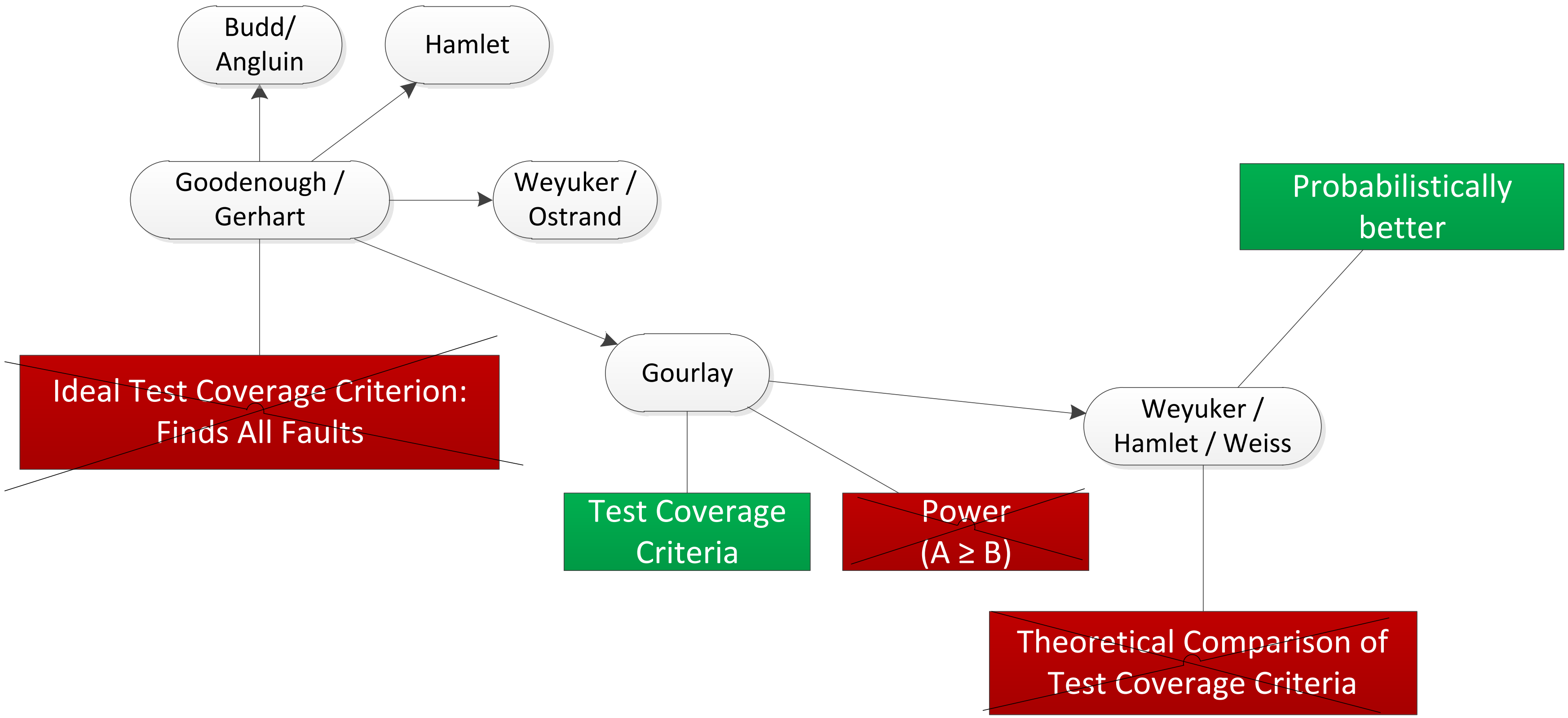


Theory of Testing - History

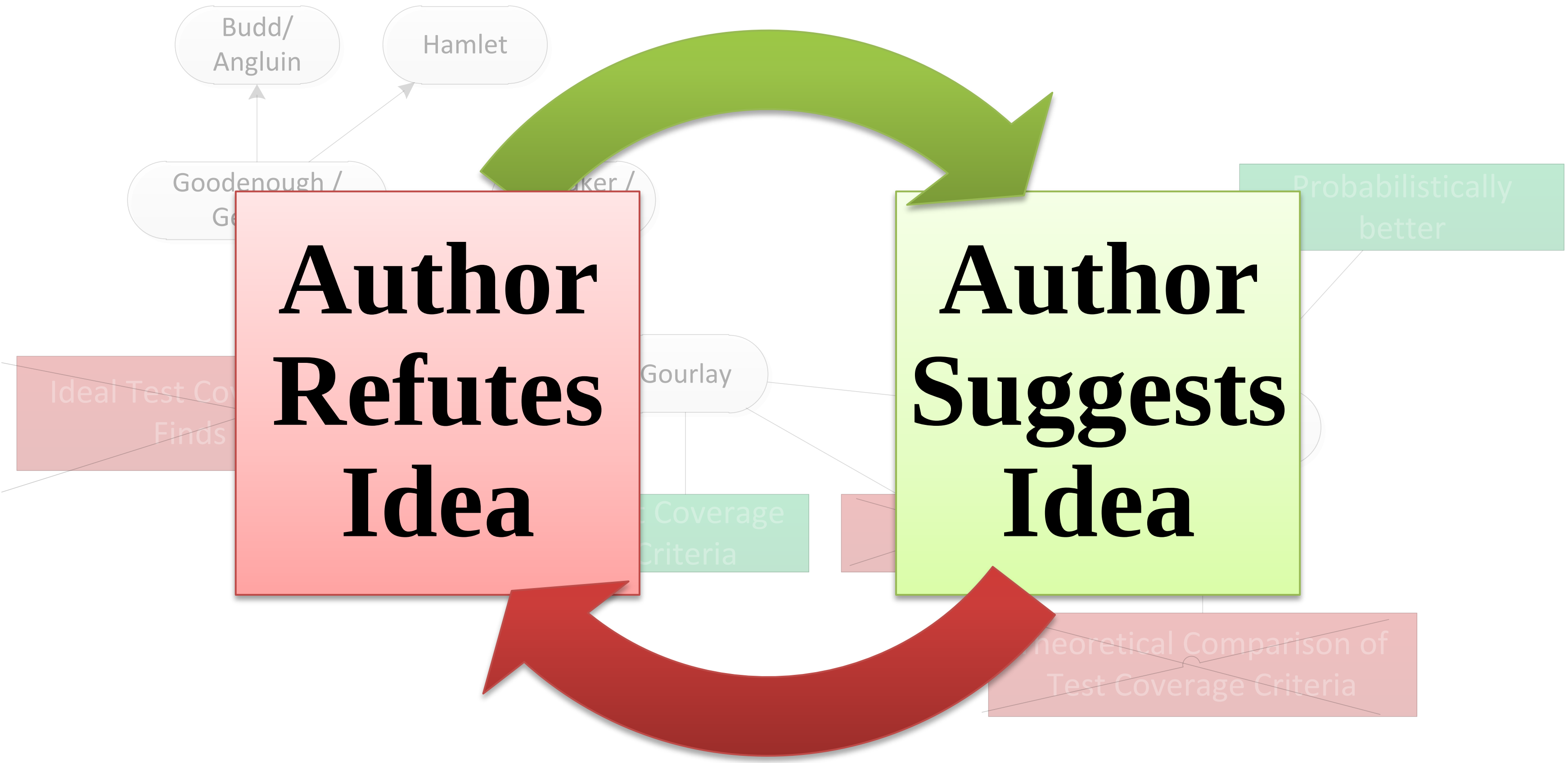


University of Minnesota
Software Engineering Center

Theory of Testing - History



Theory of Testing - History



Gourlay's Framework

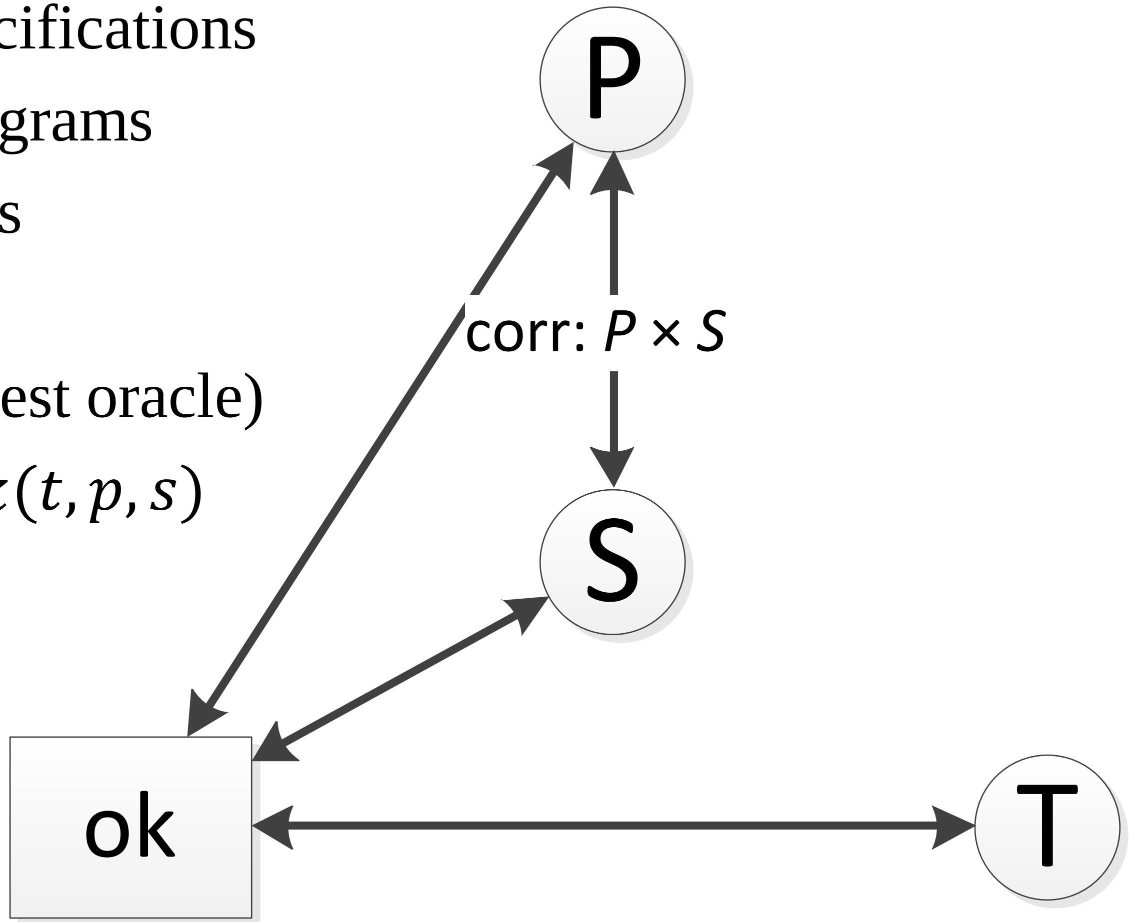
A Mathematical Framework for the Investigation of Testing

John Gourlay

IEEE Transactions on Software Engineering, 1983

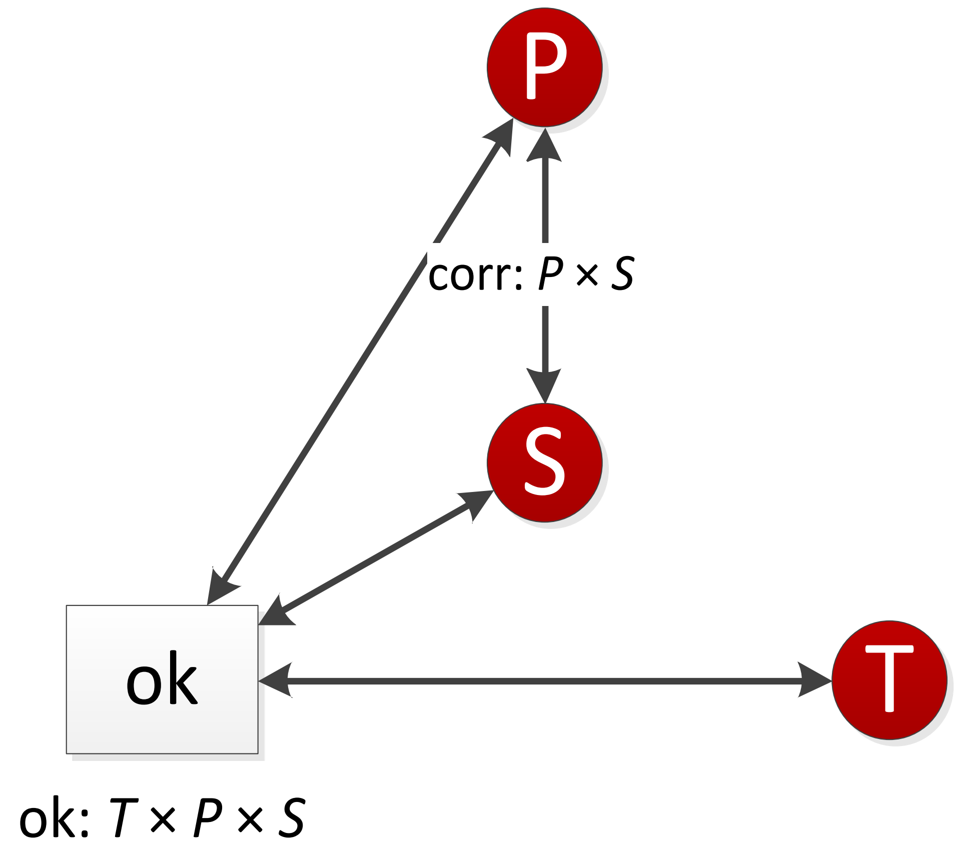
Gourlay's Framework

- S is a set of specifications
- P is a set of programs
- T is a set of tests
- $corr: P \times S$
- $ok: T \times P \times S$ (test oracle)
- $corr(p, s) \rightarrow ok(t, p, s)$



Gourlay's Framework - Problems

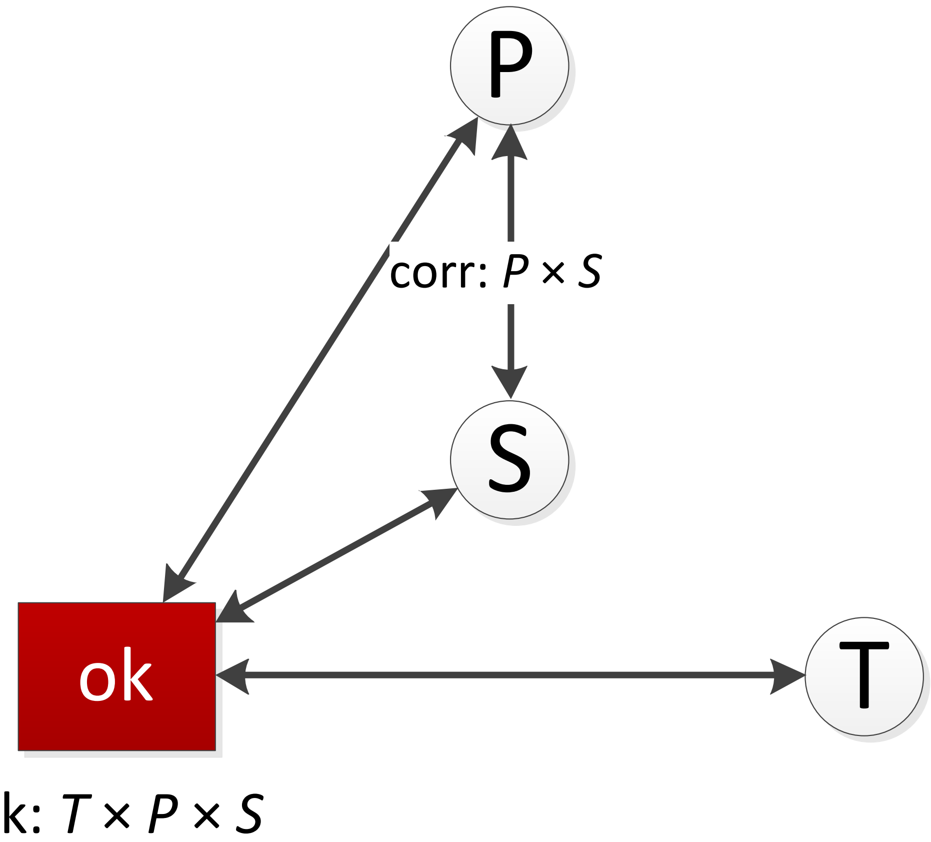
- S is a set of specifications
- P is a set of programs
- T is a set of tests
- $corr: P \times S$
- $ok: T \times P \times S$
- $corr(p, s) \rightarrow ok(t, p, s)$



Problem: no partial correctness

Gourlay's Framework - Problems

- S is a set of specifications
- P is a set of programs
- T is a set of tests
- $corr: P \times S$
- $ok: T \times P \times S$
- $corr(p, s) \rightarrow ok(t, p, s)$

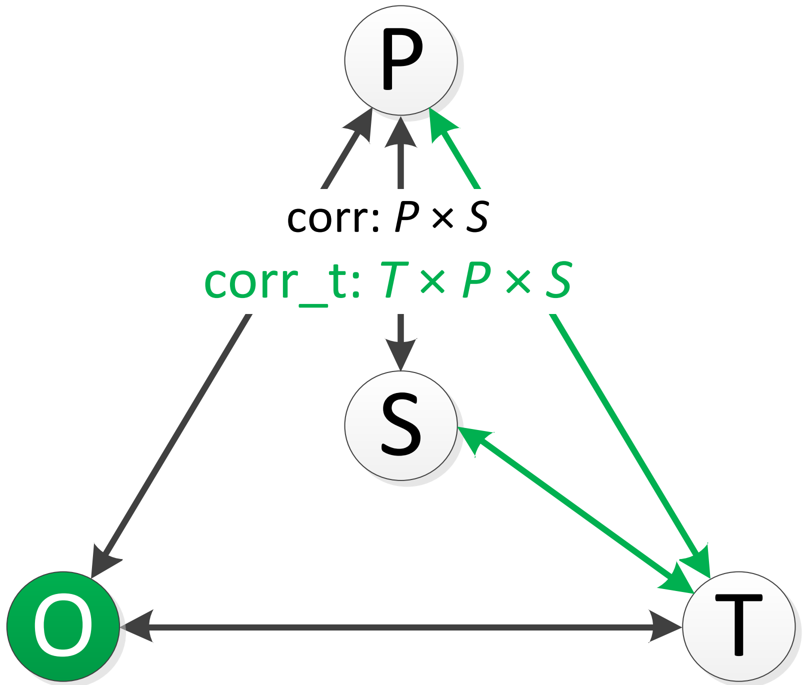


Problem: *ok* is fixed, cannot vary test oracle

Gourlay's Framework - Extension

- S is a set of specifications
- P is a set of programs
- T is a set of tests
- O is a set of test oracles
- $corr: P \times S$
- $corr_t: T \times P \times S$
- $\forall t \in T, corr_t(t,p,s) \rightarrow corr(p,s)$

Solution #1: add predicate $corr_t$



Solution #2: replace *ok* with set of predicates O, $\forall o \in O, o: T \times P$

Matt Staats, Michael W. Whalen, and Mats P.E. Heimdahl. Programs, Tests, and Oracles: The Foundations of Testing Revisited. *33rd ACM/IEEE International Conference on Software Engineering*. Honolulu, Hawaii, May, 2011. Paper awarded the ACM Distinguished Paper Award.

UNIVERSITY OF MINNESOTA
Software Engineering Center

Application of Extension

Formalize concepts related to test oracles

- Oracle relationship to correctness
  - **Complete:** $corr_t(t, p, s) \rightarrow o(t, p)$
  - **Sound:** $o(t, p) \rightarrow corr_t(t, p, s)$
  - **Precise:** $o(t, p) \leftrightarrow corr_t(t, p, s)$
- Adequacy of testing process
  - **Oracle adequacy criterion:** $O_C : P \times S \times O$
  - **Complete adequacy criterion:** $\mathrm{T}O_C : P \times S \times 2^T \times O$
- Formal oracle comparisons
  - Power comparison
  - Probabilistic comparison
- Some previous work is most likely not valid in the face of varying oracles (and program structures)

University of Minnesota
Software Engineering Center

Two Approaches

Empirical Studies

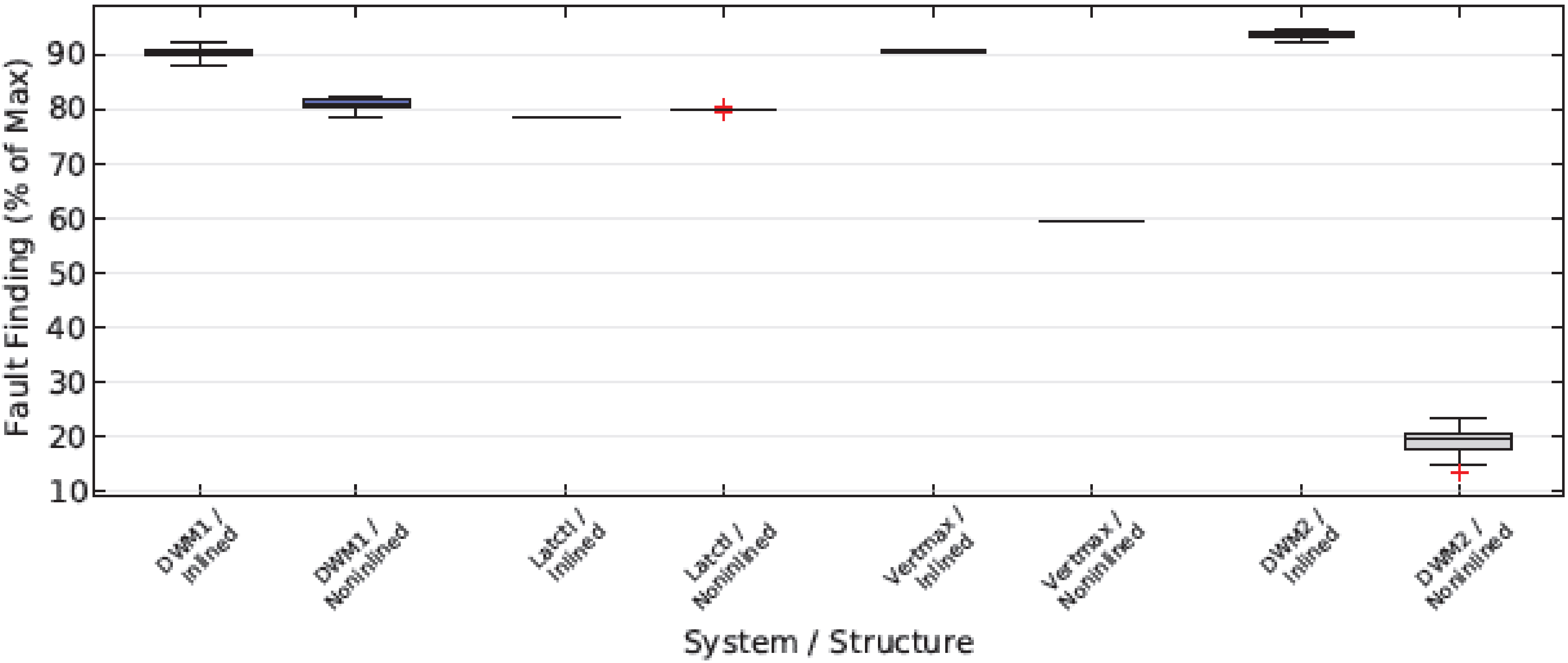
Test Metrics

- **Idea**: Measure how well tests cover the structure of code as an approximation of "goodness" of testing
  - Examples:
    - Statement coverage
    - Decision coverage
    - Modified Condition/ Decision Coverage (MC/DC)
  - Used as adequacy criteria for critical avionics software
- Are these good metrics?

- **Effective** at finding faults;
  - Better than random testing for suites of the same size
  - Better than other metrics
  - It explicitly accounts for oracle
- **Robust** to simple changes in program structure
- **Reasonable** in terms of the number of required tests and coverage analysis

There Are Weaknesses

- Program structure matters



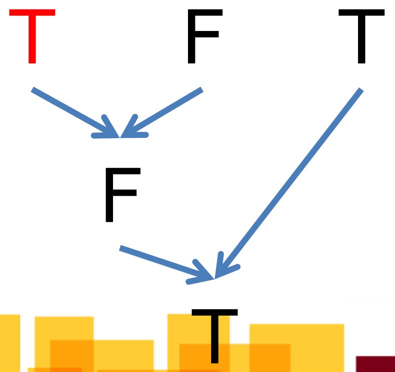
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision's outcome

a = **T**
b = F
c = T

(a && b) || c



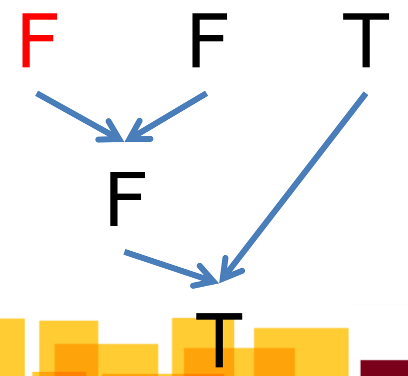
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision's outcome

a = **F**
b = F
c = T

(a && b) || c

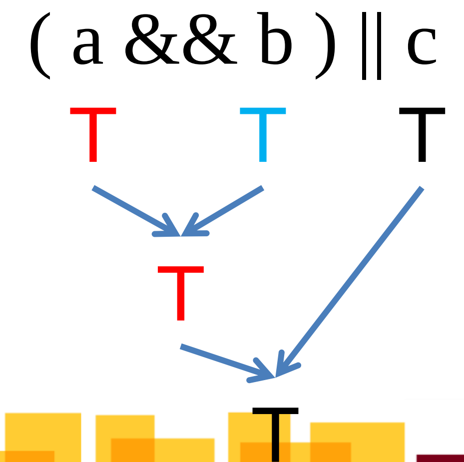


Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision's outcome

a = **T**
b = **T**
c = T



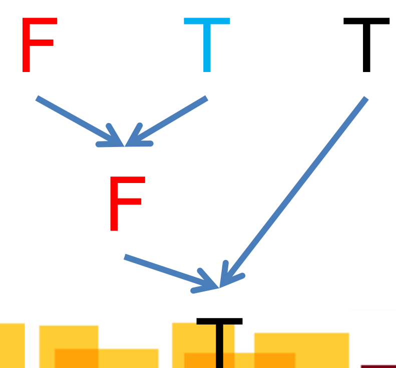
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision's outcome

a = **F**
b = **T**
c = T

(a && b) || c



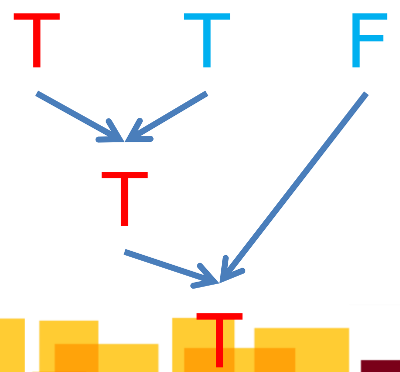
Modified Condition/Decision Coverage (MC/DC)

To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision's outcome

a = **T**
b = **T**
c = **F**

(a && b) || c



Modified Condition/Decision Coverage (MC/DC)

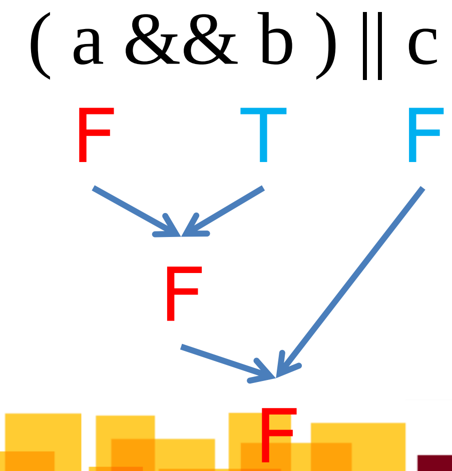
To satisfy MC/DC:

- Every basic condition in a decision in the model should take on all possible outcomes at least once, and
- Each basic condition should be shown to independently affect the decision's outcome

a = **F**

b = **T**

c = **F**



Masking and Measurement of MC/DC

Version 1:

Non-Inlined Implementation

expr1 = in1 **or** in2;

out1 = expr1 **and** in3;

Version 2:

Inlined Implementation

out1 = (in1 **or** in2) **and** in3;

Tests in green satisfy MC/DC for version 1 but not 2

In1	In2	In3	In1 or in2	(in1 or in2) and in3
F	F	F	F	F
F	F	T	F	F
F	T	F	T	F
F	T	T	T	T
T	F	F	T	F
T	F	T	T	T
T	T	F	T	F
T	T	T	T	T

Masking and Measurement of MC/DC

Version 1:

Non-Inlined Implementation

expr1 = in1 **and** in2;

out1 = expr1 **and** in3;

Version 2:

Inlined Implementation

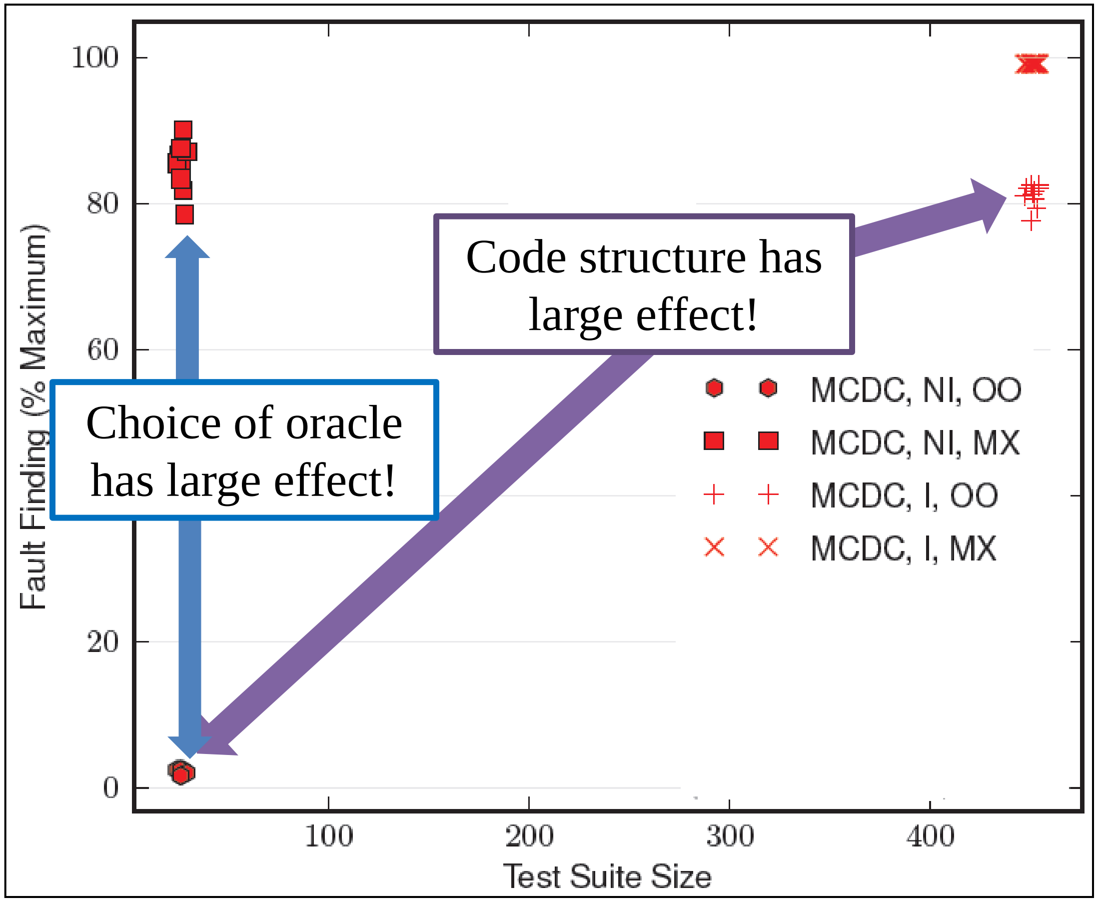
out1 = (in1 **or** in2) **and** in3;

Tests in green satisfy MC/DC for version 1 but not 2

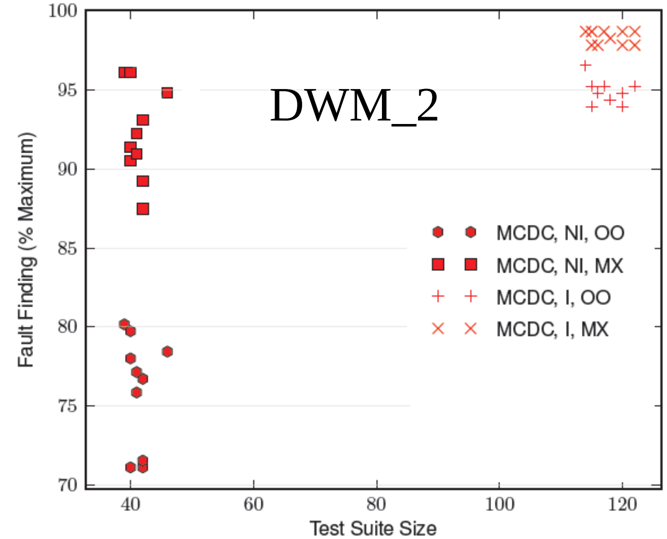
Tests still pass if we replace 'or' with 'and'

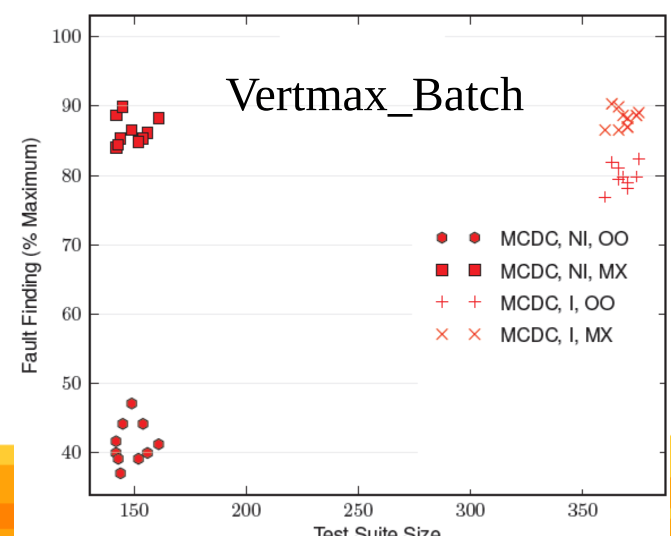
In1	In2	In3	In1 & in2	(in1 & in2) and in3
F	F	F	F	F
F	F	T	F	F
F	T	F	F	F
F	T	T	T	T
T	F	F	F	F
T	F	T	T	T
T	T	F	T	F
T	T	T	T	T

MC/DC Effectiveness



DWM_1





Another Way to Look at MC/DC

- Masking MC/DC can be expressed:

$$(D(t_i) \neq D[true/c_n](t_i)) \wedge (D(t_j) \neq D[false/c_n](t_j))$$

Where $P[v/e_n]$ means, For program *P*, the computed value for the *nth* instance of expression *e* is replaced by value *v*

- Describes whether a condition is observable in a decision (i.e., not masked)
- **Problem**: we can rewrite programs to make decisions large or small (and MC/DC easy or hard to satisfy!)

UNIVERSITY OF MINNESOTA
Software Engineering Center

Observable MC/DC

Michael W. Whalen, Gregory Gay, Dongjiang You, and Mats P.E. Heimdahl. Observable Modified Condition/Decision Coverage. Proceedings of the 35th ACM/IEEE International Conference on Software Engineering (ICSE'13). San Francisco, USA, May 2013.

Idea: lift observability from decisions to programs

- Explicitly account for oracle
- Strength should be unaffected by simple program transformations (e.g., inlining)

$$
\begin{aligned}
&(\forall c_n \in Cond(P) \ . \\
&\qquad (\exists t \in T \ . \ (P(t) \neq P[true/c_n](t))) \ \ \wedge \\
&\qquad (\exists t \in T \ . \ (P(t) \neq P[false/c_n](t))))
\end{aligned}
$$

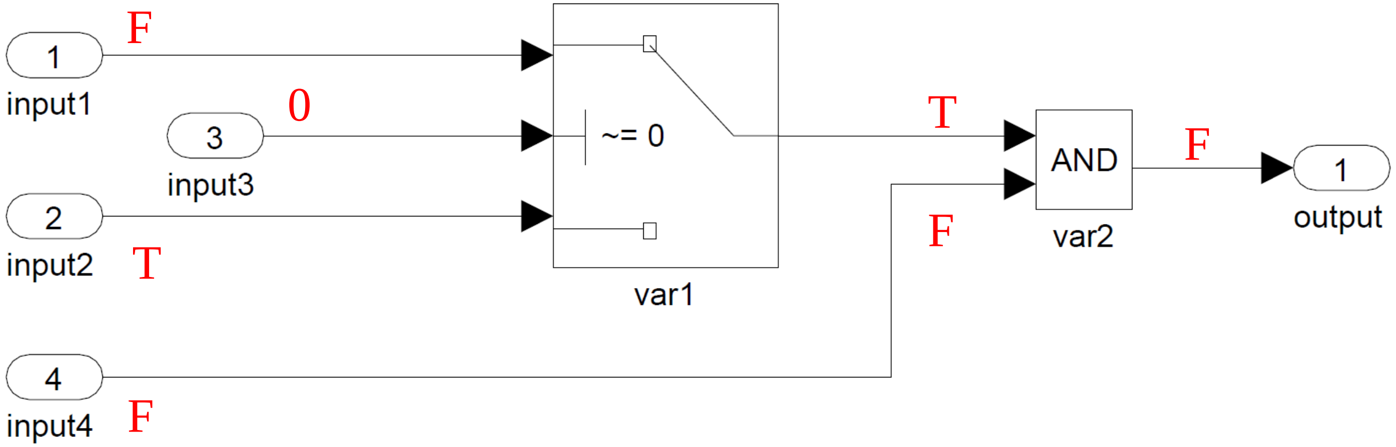
where $Cond(P)$ is the set of all conditions in program P

University of Minnesota
Software Engineering Center

Tagged Semantics

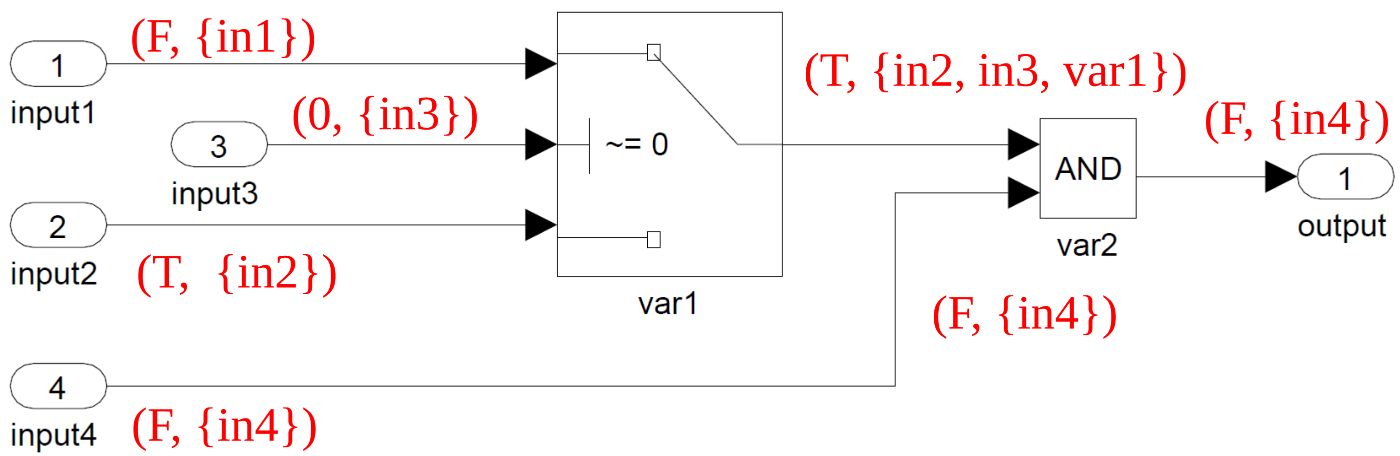
- Semantic definition is unwieldy for measurement and test generation
  - Requires separate program variant for every condition
  - Run variant in parallel with original program
- Approximate by tagging semantics
  - Assign each condition a tag
  - Track these tags through program execution (both the condition's tag and value)
  - If a tag reaches the output, the obligation is satisfied

An Example Program (in Simulink)



Does the value of input2 affect the output? **No**

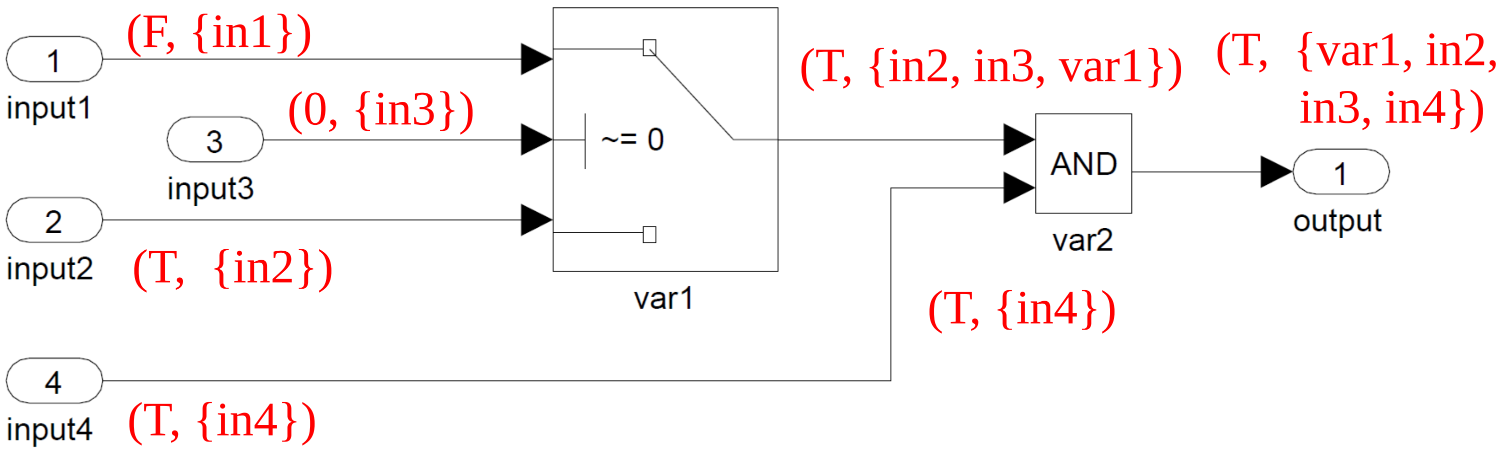
Evaluation using Tags



Does the value of input2 affect the output? **No**

University of Minnesota
Software Engineering Center

Evaluation using Tags



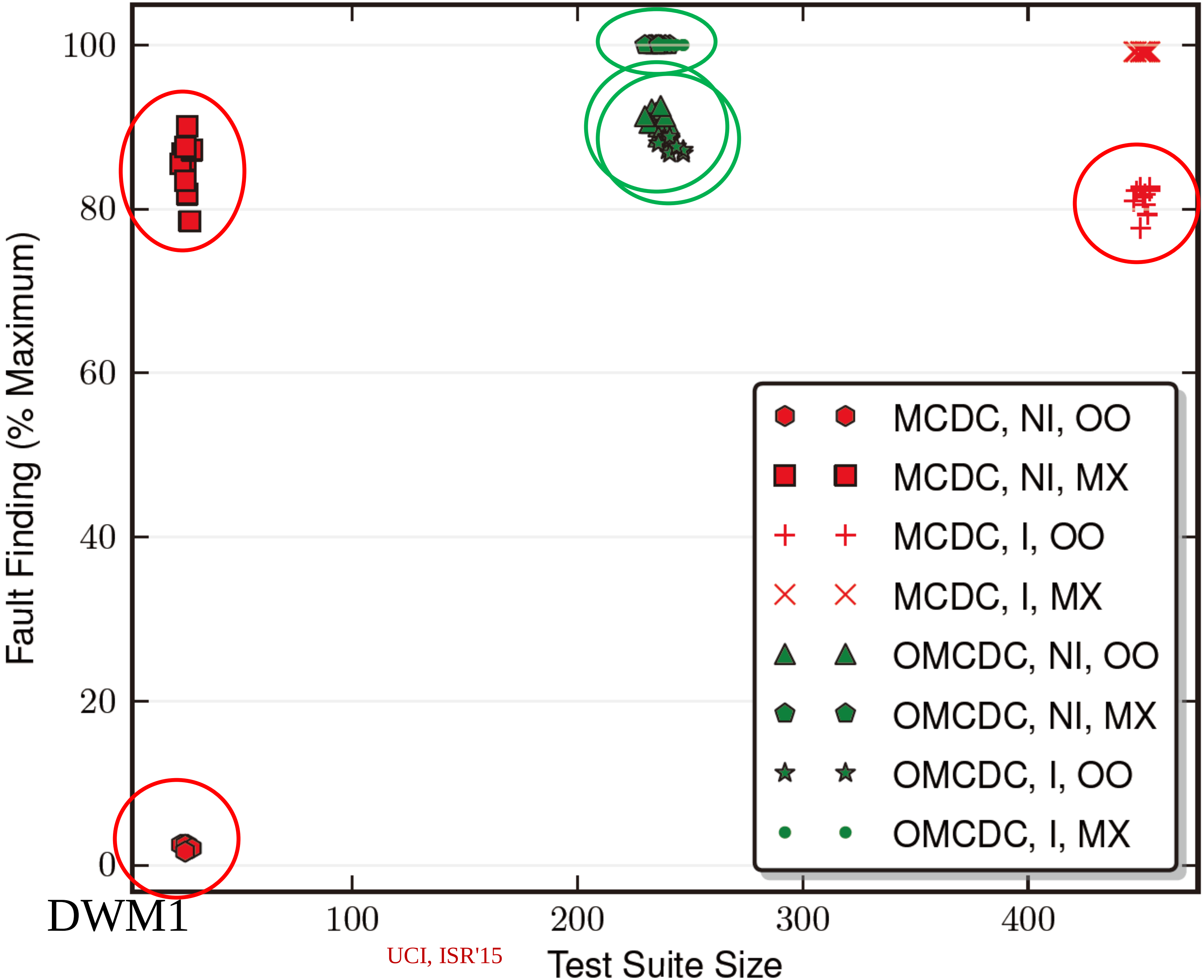
Does the TRUE value of input2 affect the output?

Yes. If input4 is **true,** then var1 is not masked out by the AND gate, so input2 propagates.

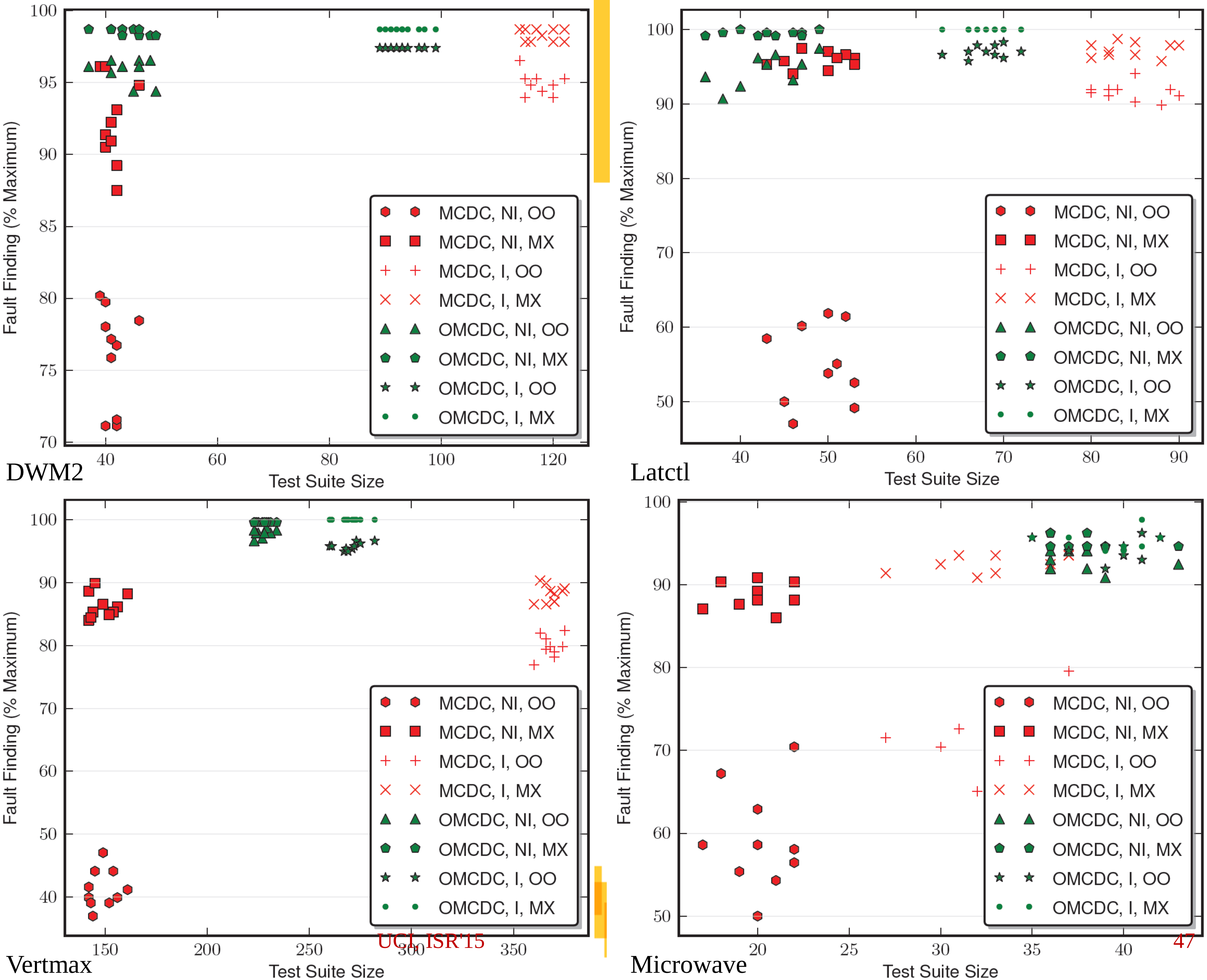
We can define the tagging semantics by instrumenting the original program; we then use this instrumented program for both test measurement and test generation.

Experiments and Evaluation

- For each of 4 industrial avionics systems and 1 toy system:
- Create inlined and non-inlined implementations
- Test suite generation
  - Counterexample-based approach guarantees maximum possible coverage (using Kind)
  - 10 test suites each for OMC/DC and MC/DC
- Mutant generation
  - 250 mutants for each case example
  - Removed functionally equivalent mutants
    - Finite systems, decidable and fast
- Output-only and maximum oracles
  - Output-only oracle compares values only for output variables
  - Maximum oracle compares values for all internal variables and outputs


100
80
60
40
20
0
Fault Finding (% Maximum)
100
200
300
400
Test Suite Size
MCDC, NI, OO
MCDC, NI, MX
MCDC, I, OO
MCDC, I, MX
OMCDC, NI, OO
OMCDC, NI, MX
OMCDC, I, OO
OMCDC, I, MX

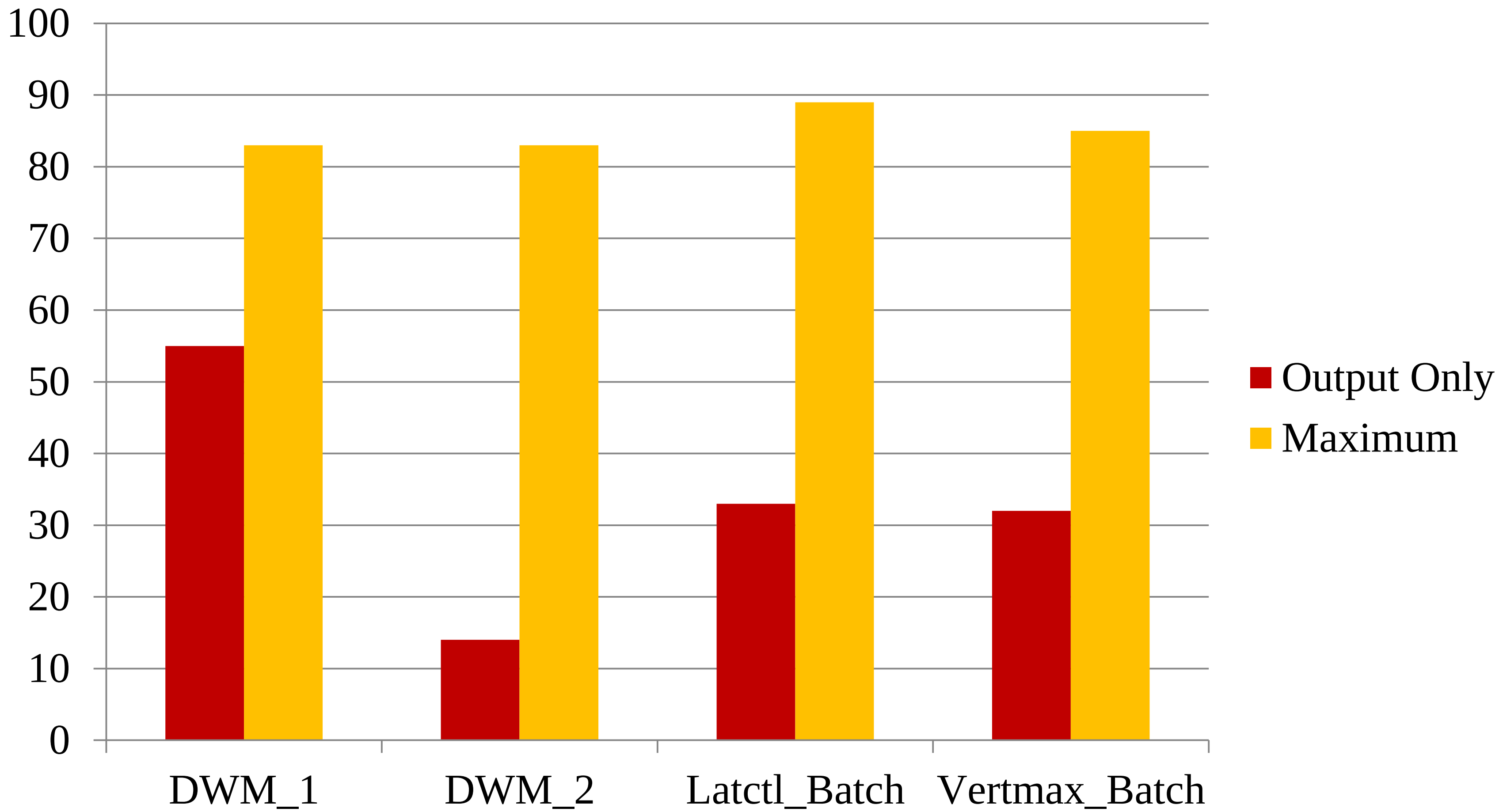
DWM1


DWM2
Latctl
Vertmax
Microwave
Fault Finding (% Maximum)
Test Suite Size
MCDC, NI, OO
MCDC, NI, MX
MCDC, I, OO
MCDC, I, MX
OMCDC, NI, OO
OMCDC, NI, MX
OMCDC, I, OO
OMCDC, I, MX

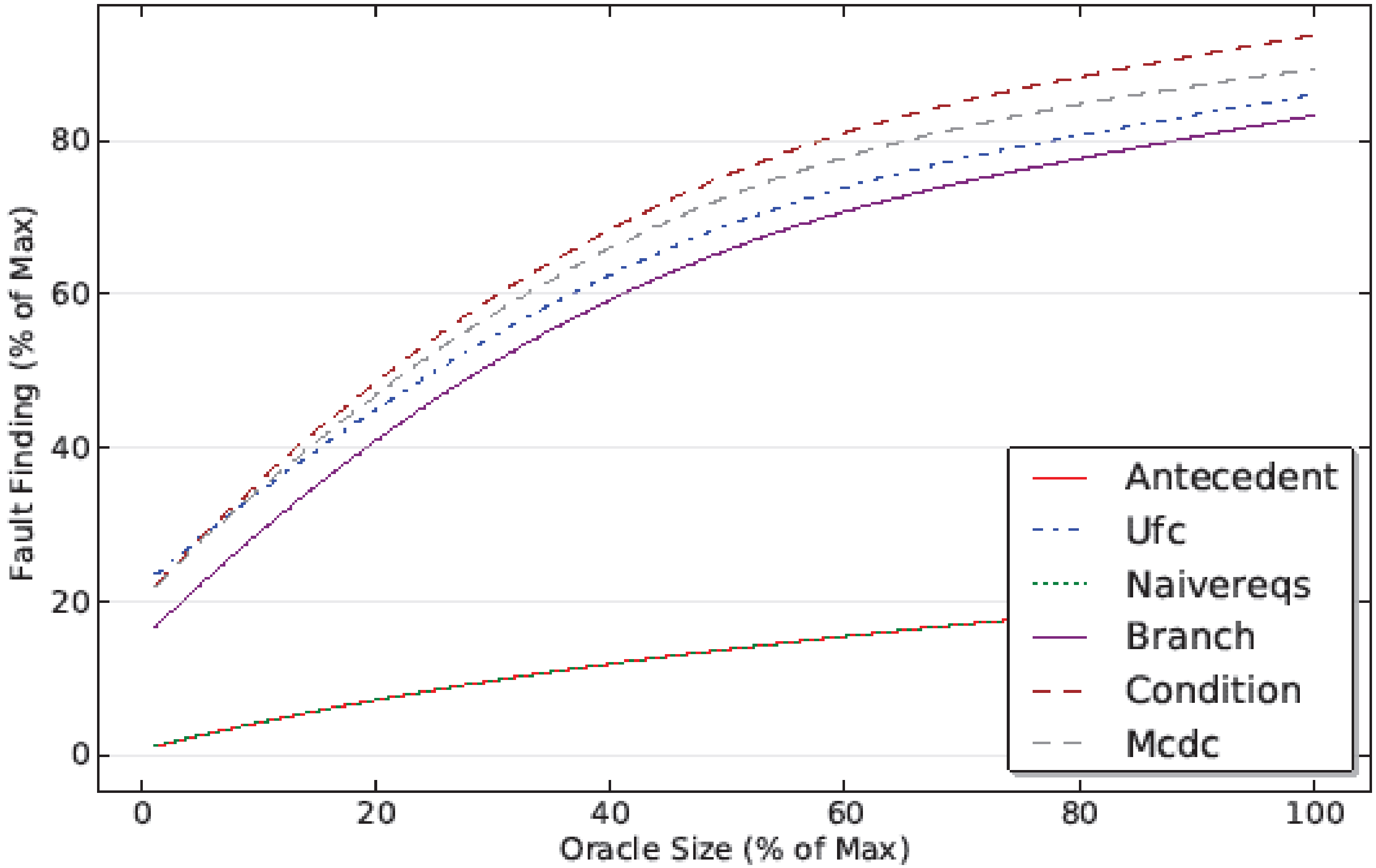
Achievable Obligations

	Structure	OMC/DC	MC/DC
DWM1	Non-Inlined	99.9%	100%
	Inlined	68.7%	98.1%
DWM2	Non-Inlined	89.8%	95.3%
	Inlined	**57.5%**	64.8%
Latctl	Non-Inlined	93.4%	100%
	Inlined	92.7%	99.6%
Vertmax	Non-Inlined	98.2%	100%
	Inlined	96.4%	99.1%
Microwave	Non-Inlined	**68.9%**	**98.9%**
	Inlined	72.2%	94.2%

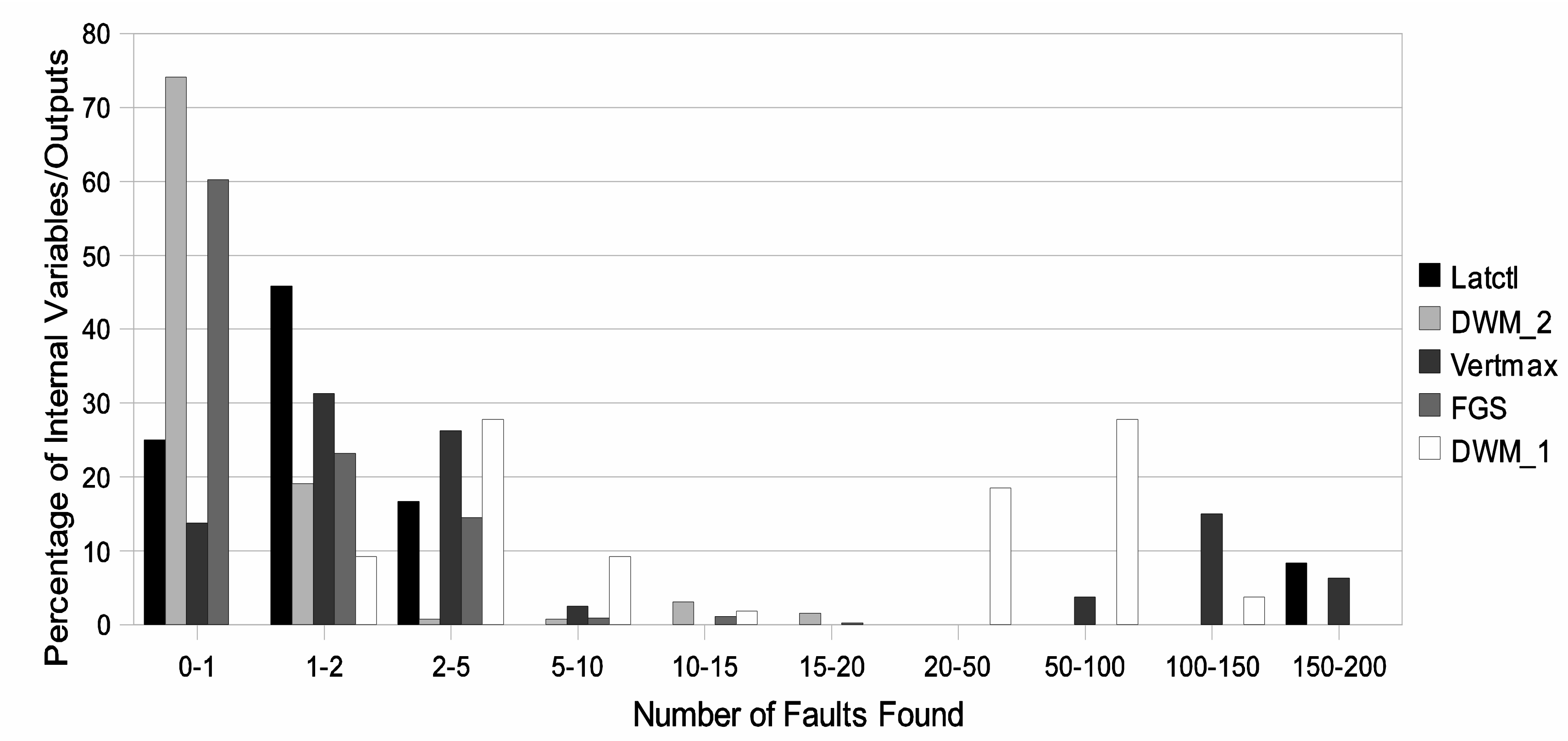
Oracle Matters



More Oracle Variables is Better

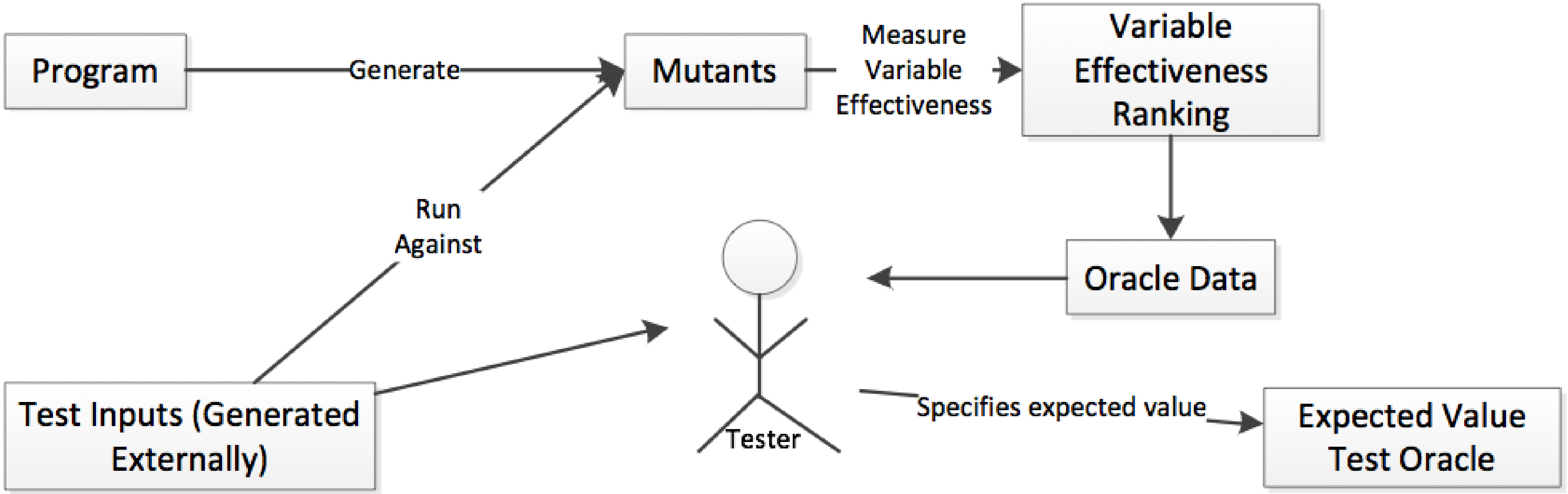


Some Variables Are Better



University of Minnesota
Software Engineering Center

Oracle Selection Process

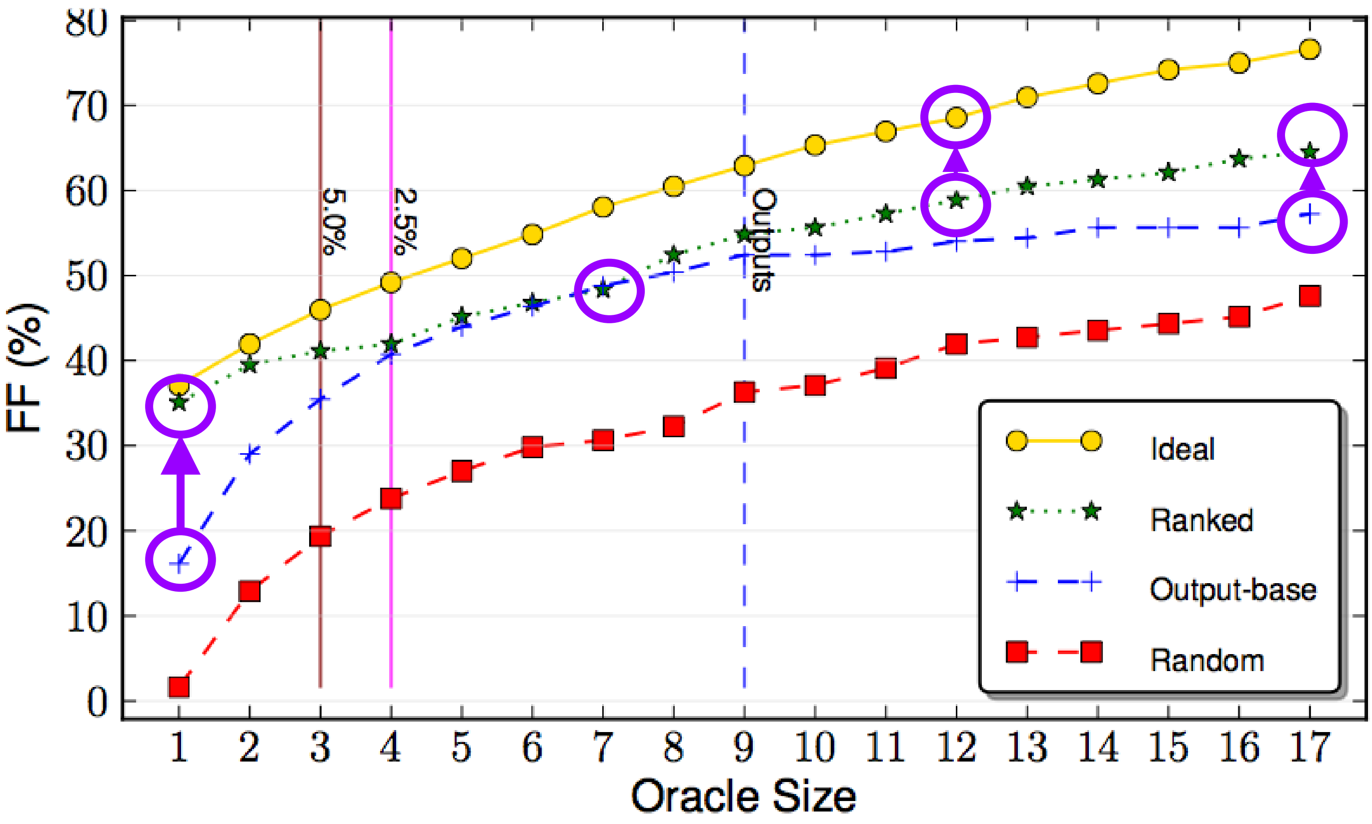


Matt Staats, Gregory Gay, and Mats P.E. Heimdahl. Automated Oracle Creation Support, or: How I Learned to Stop Worrying About Fault Propagation and Love Mutation Testing. Proceedings of the 34th ACM/IEEE International Conference on Software Engineering (ICSE'12). Zurich, Switzerland, May 2012.

University of Minnesota
Software Engineering Center

Results - Effectiveness

Common Pattern for Structure-based, Random Tests:



Summary and Future Work

- Testing effectiveness is influenced by many factors
  - Interrelationship between Program, Specification, Test Set, and Oracle
- Potential benefits in examining other artifacts in software testing
  - Can we discover "good" combinations?
- Potential dangers in adopting too narrow a view of a software testing
- **Much more work to be done!**

- Observable MC/DC
  - Robust to program structure
  - Better fault finding than MC/DC
  - Explicitly accounts for oracle
- Oracle discovery
  - Find the best variables to monitor
- Future work
  - Discover "complete" coverage criteria
    - Match program, specification, tests, and oracle in "good" ways
  - Larger studies with C and Java code
  - Dismiss uncoverable code

Questions