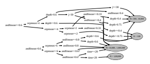
Testing Systems That Change

Tim Menzies[‡],
Bojan Cukic[†],
Harhsinder Singh[§]

[‡]NASA/WVU, 100 University Drive, Fairmont, USA
[†]Com. Sci. and Electrical Engineering, WVU,
[§]Department of Statistics, WVU

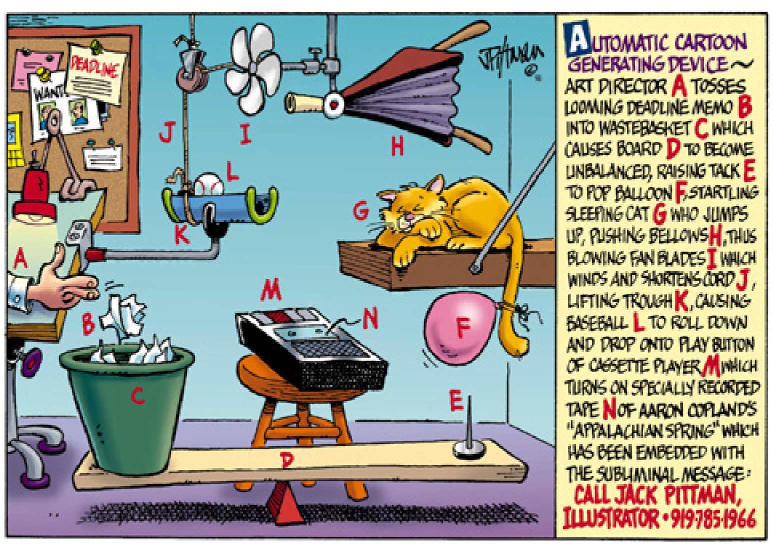
tim@menzies.com,
cukic@csee.wvu.edu
hsingh@stat.wvu.edu

August 22, 2000



Desired:

Actual:



Summary

- ✓ Testing systems that change not so hard.
- ✓ Indeterminacy not the major influence on testability.
- ✓ Adaptation followed by rapid re-test= practical
- ✓ Mutation testing not overly-complicated.
- ✓ Tim's Law: Often, a small number of random probes will yield as much information as a large number of considered probes.
- × "Test-ability" not just a static property, but...
- ✓ Can design for better testability.
- × $\neg\Box(parts = whole)$; utility of formal analysis of parts?
- × Average case analysis only for testing as reachability; not for fault localization/ fault removal of mission-critical systems.

What Systems Change?

Any working software system

- Using the hammer changes the hammer.

Adaptation via machine learning:

- Pre-launch behavior $\neq$ flight behaviour

Indeterminacy

- Same inputs, different days, different outputs.
- Seen in AI and requirements engineering.
- ”Indeterminacy is the enemy of reliability” [Levenson, 1995].

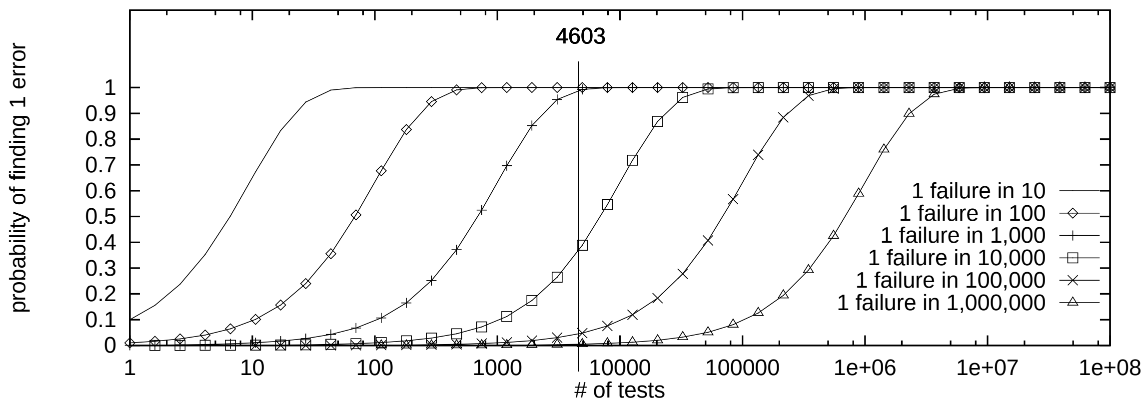
Mutation analysis

- Bash the program into another program: can you detect the changes?

IV&V: The Outsiders

- The IV&V Facility:
  - Independent assessment (quick peeks).
  - IV&V (long stares)
- Usually:
  - Process add-on, not process driver.
  - Heavily resource-bound
  - Incomplete specs (the axiom famine).
- Being cost effective is essential:
  - Are stochastic methods cheaper?
- Strangely:
  - Partial heuristic explorations effective.
  - If an exploration terminates, then errors++.
  - Jack's rule: 5 major-ish errors is enough
  - What is the cheapest way to find the 5?

Testing= Hard, Right?



- The above: massive over-estimate, blind to internal structure.

- Tim's law: Often, a small number of random probes will yield as much information as a large number of considered probes

  $$Out_i = F_{random}(random(In)_i)$$

  For small M and large N, usually,

  $$|Out_1 \wedge Out_2 \ldots Out_m| \approx |Out_1 \wedge Out_2 \ldots Out_n|$$

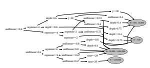
- e.g. Most program mutations give same results.
  [Budd, 1980],[Wong and Mathur, 1995], [Michael, 1997]

BTW: Black Box Probing Must be Over-Estimates

- Easy, but limited, accessibility.
  [Colomb, 1999] [Fenton and Pfleeger, 1997]

- Static analysis results:
  - Programs *much* simpler than we might think.
    [Harrold et al., 1998].
  - Few pathways within our programs.
    [Bieman and Schultz, 1992]

- Dynamic analysis results:
  - Randomized search quickly finds as much as considered search. [Selman et al., 1992].
  - Exploring all conflicts tells you little more than exploring a few: [Williams and Nayak, 1996] and
    [Crawford and Baker, 1994] (see below),
    [Menzies et al., 1999] (see below).



ISAMP

[Crawford and Baker, 1994]

```
for TRIES := 1 to MAX-TRIES
{set all vars to unassigned;
 loop
  {if everything assigned
   then return(assignments);
   pick any var v;
   v :=  random assignment;
   forwardChain();
   if contradiction exit loop;
  }
 } return failure
```

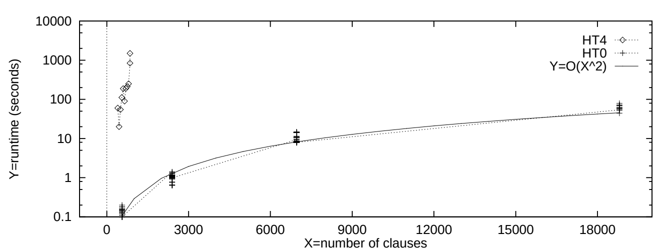
	TABLEAU: full search		ISAMP: partial, random search		
	% Success	Time (sec)	% Success	Time (sec)	Tries
A	90	255.4	100	10	7
B	100	104.8	100	13	15
C	70	79.2	100	11	13
D	100	90.6	100	21	45
E	80	66.3	100	19	52
F	100	81.7	100	68	252

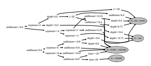
HT0

[Menzies and Michael, 1999]

```
X ror  Y :-
    maybe ->(X;Y);(Y;X).

X rand Y :-
    maybe ->(X,Y);(Y,X).

maybe :- 0 is random(2).

:- ht0(5,[1/sad,1/rich]).

ht0(0,_) :-!.
ht0(N0,G0) :-
    rememberBestCover,
    retractall(a(_,_,_)),
    sort(G0, G1),
    maplist(prove,G1,G),
    N is N0 - 1,
    ht0(N,G).

prove(In/Goal,Out/Goal):-
    delta(X),
    (call(Goal)
    -> Out is In + X
    ;  Out is In - X).

delta(X) :-
    X is 1 +
       random(10^3)/10^6.
```

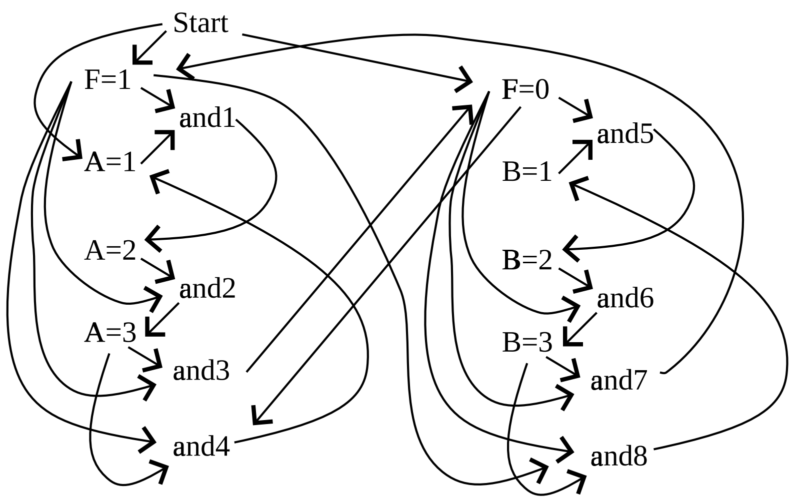
```
A of O is New :-
    a(A,O,Old),
    !,
    Old = New.
A of O is New :-
    assert(a(A,O,New)).
A of O is New :-
    retract(a(A,O,New)),
    fail.
```

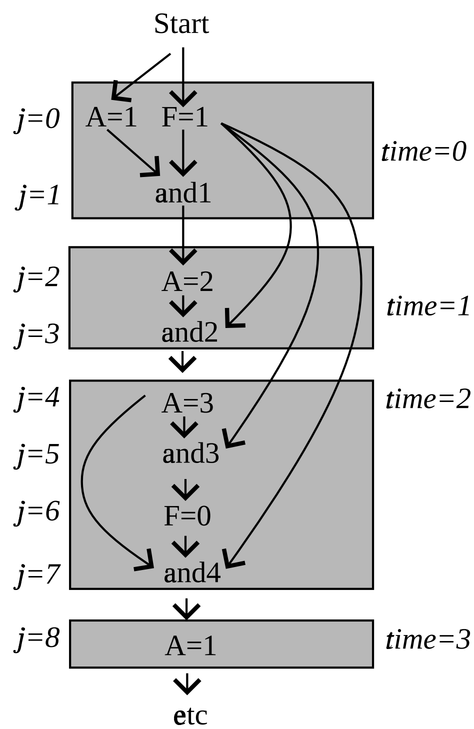


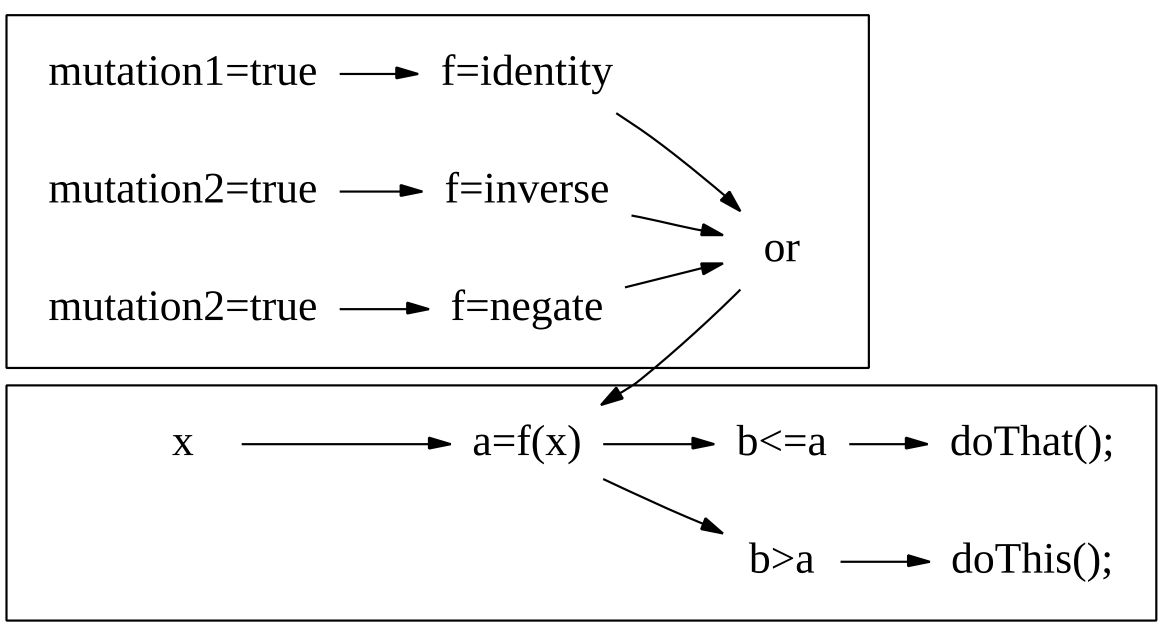


```
byte a=1;
byte b=1;
bit f=1;

active proctype A(){
 do
 :: f==1 -> if
             ::a==1 -> a=2;
             ::a==2 -> a=3;
             ::a==3 -> f=0;
                       a=1;
             fi
 od
}
active proctype B(){
 do
 :: f==0 -> if
             ::b==1 -> b=2;
             ::b==2 -> b=3;
             ::b==3 -> f=1;
                       b=1;
            fi
 od
}
```

No-edges: pre-conditions for indeterminacy;
e.g. `no(X=V1,X=V2) :- not(V1 = V2).`

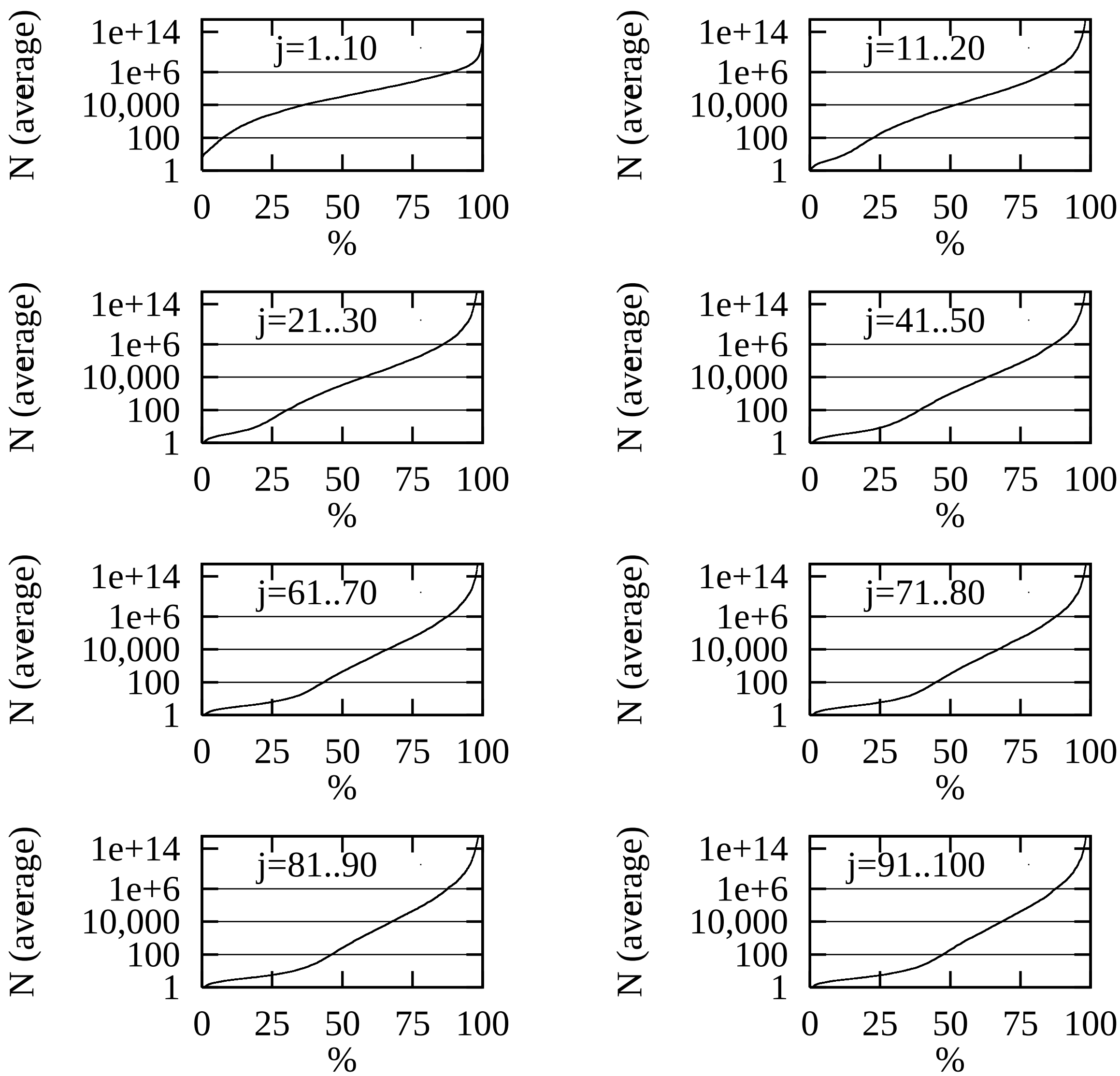


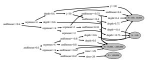




Simulation Results

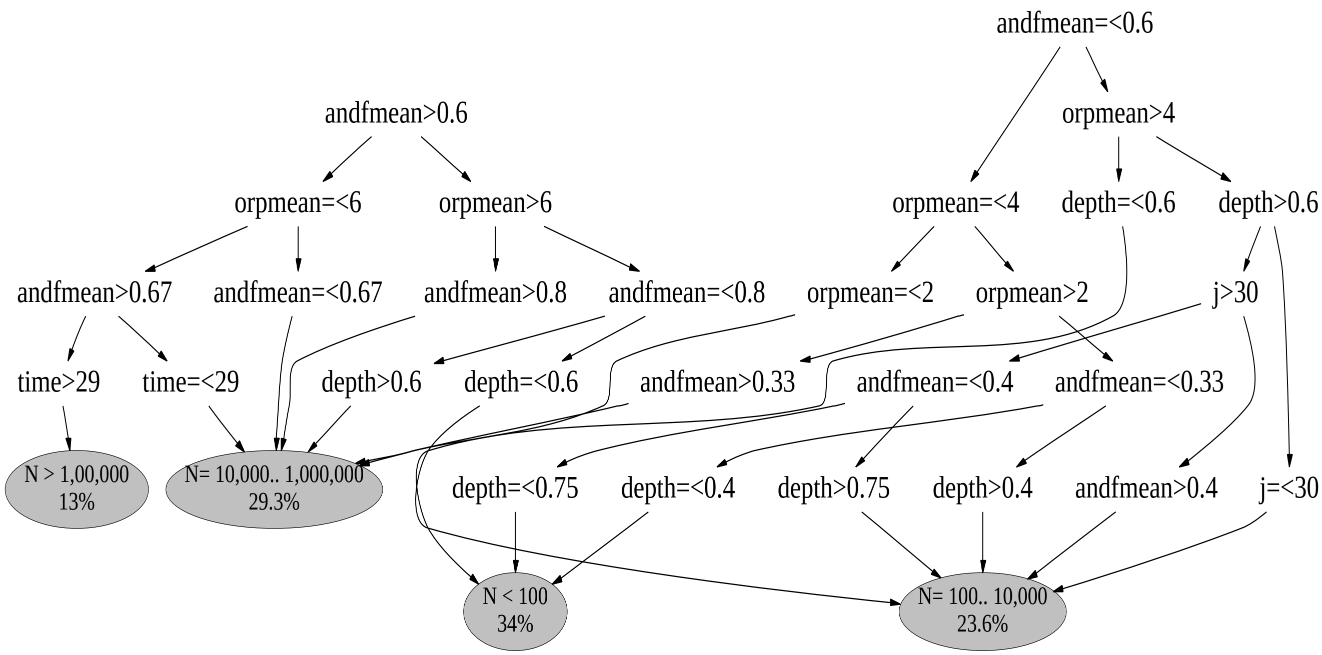
$$\begin{aligned} V &\in 500, 1500, 2500, ..10^6 \\ T &\in 1, 2, 3, ...10^2 \\ in &\in 1, 6, 11, ...10^3 \\ no_\mu &\in 0, 1, ... \end{aligned}$$





What Effects Testability?

Learnt via machine learning: C4.5 [Quinlan, 1986].

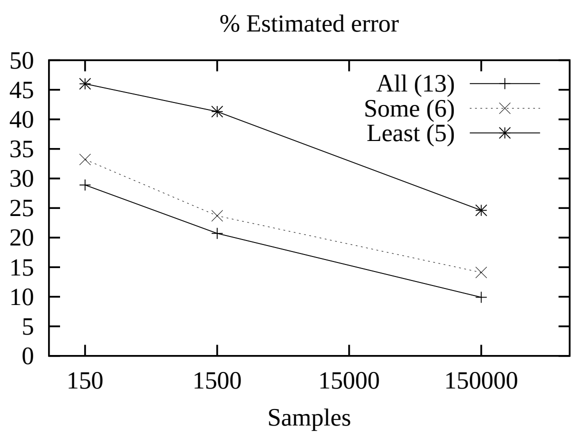


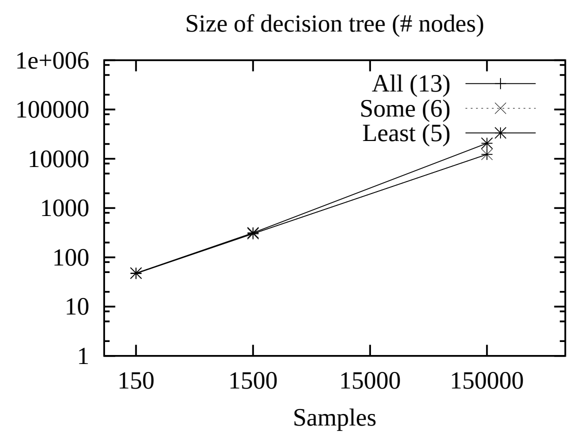
(m=45, cases=1500, estimated error=37.3%)

Note: when components combined to an aggregate, must re-calc these figures.

Sensitivity Experiments

		Group		
		All	Some	Least
$time$	time ticks	√	√	√
j	height	√	√	√
$depth$	mean relative height of parents (β)	√	√	√
orp_μ	$\gamma(orp)$ mean	√	√	√
$andp_\mu$	$\gamma(andp)$ mean	√	√	√
$andf_\mu$	mean and node frequency (β)	√	√	
$andp_\alpha$	$\gamma(andp)$ skew	√		
orp_α	$\gamma(orp)$ skew	√		
no_α	$\gamma(no)$ skew	√		
no_μ	$\gamma(no)$ mean	√		
in	number of inputs	√		
v	number of nodes	√		
$isTree?$	0,1	√		
classes	$1 \ldots 10^2$ or $10^2 \ldots 10^4$ or $10^4 \ldots 10^6$ or $10^6 \ldots \infty$	√	√	√





1) Cost of ignoring skews, no, program size, size of inputs $< 5\%$.
2) Assessing testability may ignore indeterminacy, size of inputs.
3) Assessing testability needs dynamic data ($depth$).

Conclusions

✓ Testing systems that change not so hard.

✓ Indeterminacy not the major influence on testability.

✓ Adaptation followed by rapid re-test= practical

✓ Mutation testing not overly-complicated.

✓ Tim's Law: Often, a small number of random probes will yield as much information as a large number of considered probes.

× "Test-ability" not just a static property, but...

✓ Can design for better testability (ish).

- On any execution, update stats on $< \#runs, time, j, depth, andf_{\mu}, orp_{\mu}, andp_{\mu} >$.
- Pass this *testing signature* to anyone who requests it.

× Utility of formal analysis of components questionable.

× Average case analysis only for testing as reachability; not for fault localization/ fault removal of mission-critical systems.

Discussion

1. “Would not considered reflection (e.g. over a formal model) be a better strategy than random guessing?”

2. “More details on the maths?”

3. “For mission critical software, is an average case analysis adequate?”

. . .

NAYO Graph Parameters

$$
\begin{aligned}
P_{av} &= \frac{\sum_{j=0}^{jMax} P[j]}{jMax} \\
P[j] &= andf[j] * P[j]_{and} + orf[j] * P[j]_{or} \\
P[j]_{and} &= \prod_{1}^{andp[j]} P[i] \\
P[j]_{or} &= \left(1 - \prod_{1}^{orp[j]} (1 - P[i]) \right) * \ldots \\
P[0]_{or} &= \frac{in}{V * T} \\
P[0]_{and} &= 0 \\
j &= height \\
i &= \beta(depth) * (j - 1) \\
\mu, \alpha &= mean, skew \\
andp_{\mu}, andp_{\alpha} &= and\, parents \\
orp_{\mu}, orp_{\alpha} &= or\, parents \\
no_{\mu}, no_{\alpha} &= \frac{no\ edges}{or\ node} \\
orp[j], andp[j] &= \gamma \left(\alpha, \frac{\mu}{\alpha} \right) \\
andf[j] &= \beta(andf_{\mu}) \\
orf[j] &= 1 - andf[j]
\end{aligned}
$$

100,000 runs

$$
\begin{aligned}
jMax &= 100 \\
V &\in 500, 1500, 2500, ..10^6 \\
T &\in 1, 2, 3, ...10^2 \\
in &\in 1, 6, 11, ...10^3 \\
andp_\alpha, andp_\mu, & \\
orp_\alpha, no_\alpha &\in 2, 3, 4, ...18 \\
orp_\mu &\in 1, 2, 3, 4, ...10 \\
no_\mu &\in 0, 1, 2, 3, 4 \\
depth, andf_\mu &\in 0.1, 0.2, 0.3, ...0.9
\end{aligned}
$$

$$
\begin{aligned}
p(x, N) &= 1 - \left((1 - x)^N\right) \\
N(p, x) &= log(1 - p)/log(1 - x)
\end{aligned}
$$

$$
\begin{aligned}
P_{av} &= \frac{\sum_{j=0}^{jMax} P[j]}{jMax} \\
N_{av} &= N(0.99, P_{av})
\end{aligned}
$$

Classification	Threshold	%
fast and cheap	$N_{av} < 10^2$	36
fast and moderately expensive	$N_{av} < 10^4$	19
slow and expensive	$N_{av} < 10^6$	23
impossible	$N_{av} \geq 10^6$	20

Bibliography

Bieman, J. and Schultz, J. (1992). An empirical evaluation (and specification) of the all-du-paths testing criterion. *Software Engineering Journal*, 7(1):43–51.

Budd, T. (1980). *Mutation analysis of programs test data*. PhD thesis, Yale University.

Colomb, R. (1999). Representation of propositional expert systems as partial functions. Artificial Intelligence (to appear). Available from `http://www.csee.uq.edu.au/~colomb/PartialFunctions.html`.

Crawford, J. and Baker, A. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*.

Fenton, N. E. and Pfleeger, S. (1997). *Software Metrics: A Rigorous & Practical Approach*. International Thompson Press.

Harrold, M., Jones, J., and Rothermel, G. (1998). Empirical studies of control dependence graph size for c programs. *Empirical Software Engineering*, 3:203–211.

Levenson, N. (1995). *Safeware System Safety And Computers*. Addison-Wesley.

Menzies, T., Easterbrook, S., Nuseibeh, B., and Waugh, S. (1999). An empirical investigation of multiple viewpoint reasoning in requirements engineering. In *RE '99*. Available from `http://research.ivv.nasa.gov/docs/techreports/1999/NASA-IVV-99-009.pdf`.

Menzies, T. and Michael, C. (1999). Fewer slices of pie: Optimising mutation testing via abduction. In *SEKE '99, June 17-19, Kaiserslautern, Germany. Available from `http://research.ivv.nasa.gov/docs/techreports/1999/NASA-IVV-99-007.pdf`*.

Michael, C. (1997). On the uniformity of error propagation in software. In *Proceedings of the 12th Annual Confererence on Computer Assurance (COMPASS '97) Gaithersburg, MD*.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *AAAI '92*, pages 440–446.

Williams, B. and Nayak, P. (1996). A model-based approach to reactive self-configuring systems. In *Proceedings, AAAI '96*, pages 971–978.

Wong, W. and Mathur, A. (1995). Reducing the cost of mutation testing: An empirical study. *The Journal of Systems and Software*, 31(3):185–196.