# On the value of stochastic analysis for software engineering

Tim Menzies
**West Virginia University**
tim@menzies.us

**Download these slides from**
**http://now.unbox.org/all/trunk/doc/09/nicta-jul09.pdf (10 mb)**

# Sound bites

- "Random" is not "crazy"
  - It's a random world.
  - Accept it.
  - Exploit it
- This talk: arguing for "stochastics" via case studies

# Stochastics: from the Greek "Στόχος" for "aim" or "guess"

**Definition**
(from the Greek):
**Stochastics is the process of finding the underlying pattern in what appears to be random, or chaotic.**

# Problem

- 21st century software problems characterized by
  - larger assemblies
  - less knowledge of the parts
    - Less “green field” coding
    - Mash-ups
    - Out-sourcing
    - Sub-sub-contractors
    - Open-source
- Can we reason about such system?
  - Is anything certain amidst all the possibilities?
  - Can such reasoning scale to large systems?

# Solution: stochastic stability

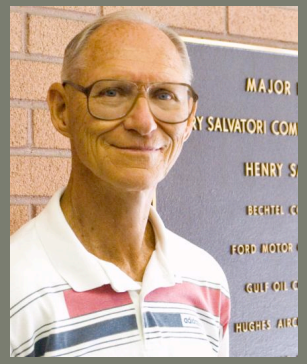
- Stochastic algorithms can sample space of uncertainty
  - Seek islands of certainty within the ocean of possibilities
- Randomized algorithm simpler, faster
- Stochastic <u>scales</u>
  - Levers the cloud computing advantage

# Structure of this talk

- Much research advocates stochastics
  - Bayesians
  - Long-tailed distributions in backtracking
  - phase transition effects,
  - log-normal distribution of reachable states
  - Keys/ collars/ clumps
  - Etc
- To many audiences, such arguments fall on deaf ears
  - This talk has offered practical examples of utility of stochastic analysis for SE
  - Each example is a counter-example to the standard anti-stochastic arguments.

1. "No value added by random thrashing"
2. "Hard to explain all that random thrash"
3. "Considered reflection is better"
4. "Non-determinism is the enemy of reliability"

# “1. No value added by random thrashing?”



USC

STAR, NOVA

(with Barry Boehm and Oussama El-Rawas)

[ASE07, ICSE09, ASE09]

random search over software process models


WVU

# Software processing modeling: the usual story

Step #1: Initialize a model
- From domain intuitions
- From historical data

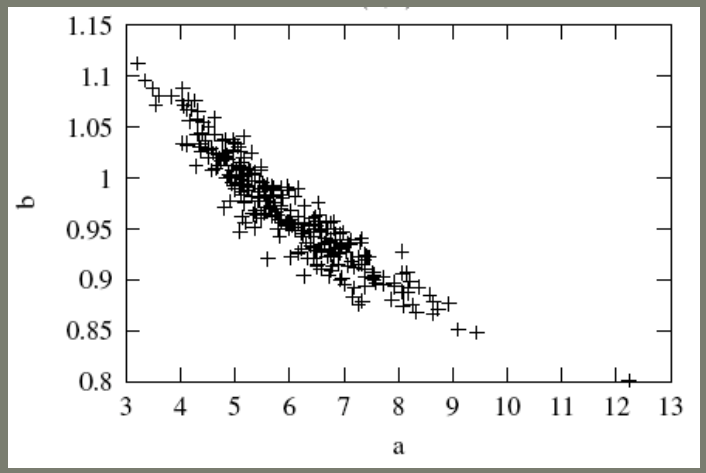
Step #2: Tune the model
- To local data

Step #3: Use the model
- Conduct trade-off studies to evaluate X vs Y

Step #2 problematic
- Can we use imported data? [Kitchenham07, TSE] [Zimmerman09, FSE]
- Is there "enough" local data?
  - "Enough" to generate stable tunings?

Effort = a * KLOC ^ b * $EM_i$

1000 timesRepeat: { Learn <a,b> from a 90% sample }




- Hard to optimize via (say) gradient descent
  - when the gradients are so uncertain

# Software process modeling: the stochastic story

- E = Estimate = model(P, T)
- P= project options
- T = space of possible tunings

- Tuning options:
  - Review 20 years of the COCOMO literature
  - Find range of tunings ever seen

- What happens if we leave tunings unconstrained?
  - And seek subsets of the project options that improve estimates…
  - … despite tuning variance

| | ranges: | min | max | fixed | |
|---|---|---|---|---|---|
| OSP: Orbital space plane | prec | 1 | 2 | data | 3 |
| | flex | 2 | 5 | pvol | 2 |
| | resl | 1 | 3 | rely | 5 |
| | team | 2 | 3 | pcap | 3 |
| | pmat | 1 | 4 | plex | 3 |
| | stor | 3 | 5 | site | 3 |
| | ruse | 2 | 4 | | |
| | docu | 2 | 4 | | |
| | acap | 2 | 3 | | |
| | pcon | 2 | 3 | | |
| | apex | 2 | 3 | | |
| | ltex | 2 | 4 | | |
| | tool | 2 | 3 | | |
| | sced | 1 | 3 | | |
| | cplx | 5 | 6 | | |
| | KSLOC | 75 | 125 | | |

Controllability assumption

- Try a stochastic search through subsets of "P"
  - Test those subsets on random sample of "T"

# Random sampling of tuning options

$Em_i = m_i x_i + b_i$

$Em_i = 1$ when $x_i$ when $=3$

$$\forall x \in \{1..6\}\ EM_i = m_a(x-3)+1$$

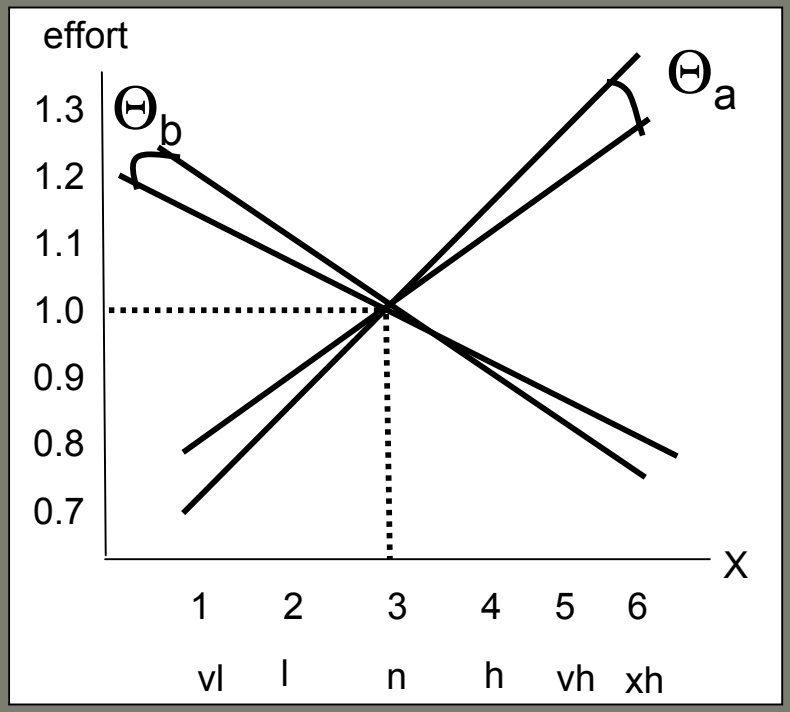
$$(0.073 \le m_a^+ \le 0.21) \wedge (-0.178 \le m_a^- \le -0.078)$$

Increase effort

cplx, data, docu
pvol, rely, ruse,
stor, time

decrease effort

acap, apex, ltex, pcap,
pcon, plex,sced,
site,toool

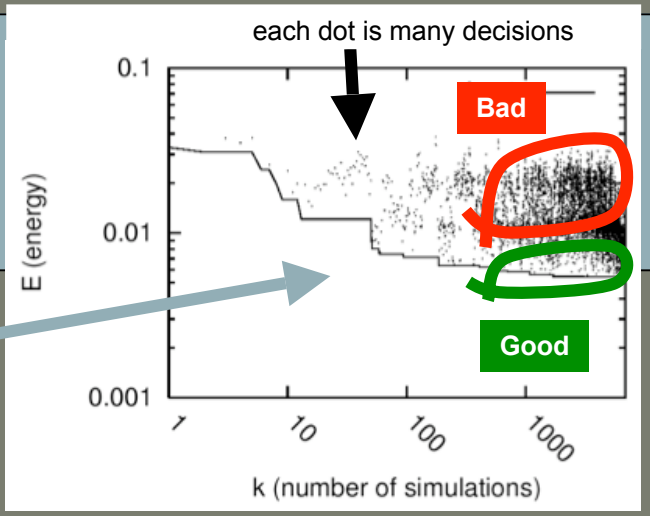


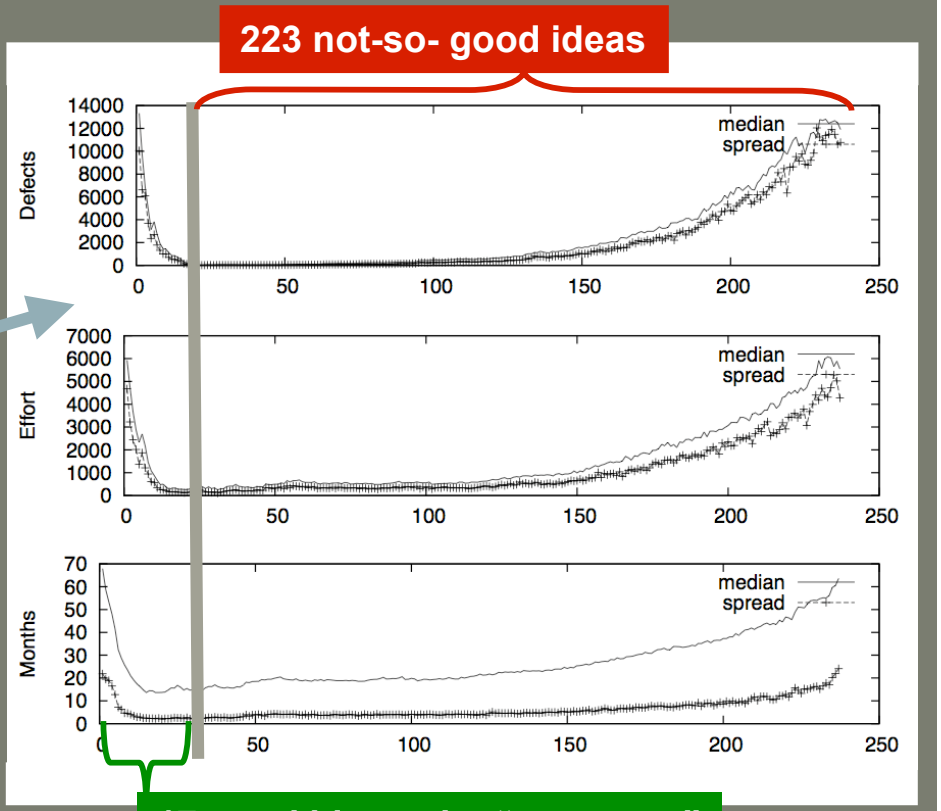

- And we have other sample tricks for the rest of
  - COCOCMO
  - COQUALMO

# Random sampling of project options

1. Simulated annealing
   - "current" = a project
   - scored using any tunings and
     $$E = \left(\sqrt{\overline{Ef}^2 + \overline{De}^2 + \overline{Th}^2}\right)/\sqrt{3}$$
   - "next" = 33% mutation of "current"
   - current = next if new score better than old
   - Else, at prob P(T), current = next
2. Rank decisions 1.. all by "goodness"
3. Experiment with setting decisions 1 <= i <= all in rank order
   - Experiment = 100 sims with random tunings
   - Median, spread = (50, 75-50)% percentile

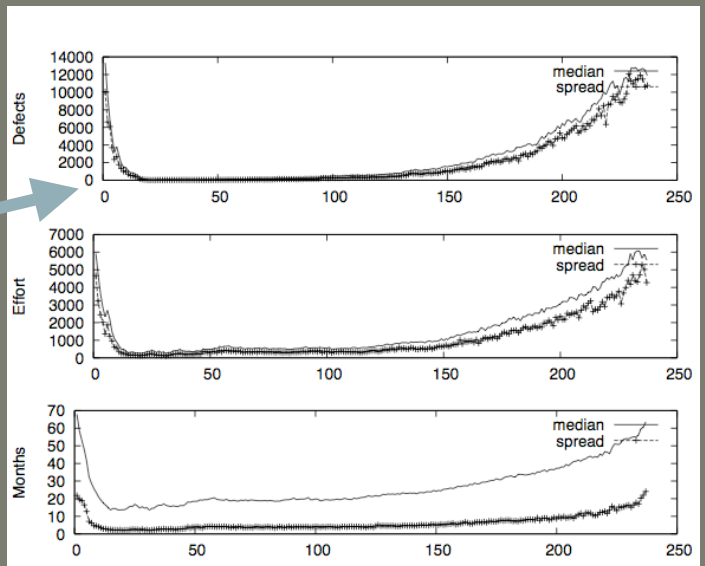
- "Policy point" : smallest i with lowest E
- Best project options are to the left of the policy point



# Discussion

- Conclusions stable across space of unknowns
- Precise tunings not needed to rank process options.
  - (And if you want actual estimates, see [Keung08])
- Succinct report: the “one slide” rule
- Now have a principled, defensible, documentable method of debating project changes.
- The above analysis assumed
  - Models where estimates are dominated by
    - project options,
    - not tuning options
  - Which is true for the USC COCOMO-family of process models [ICSP’08]
  - Not all process models have this property
    - Should it be a selection criteria for process models?



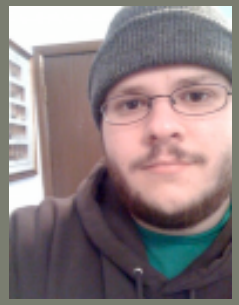

| X | feature = range |
|---|---|
| 1 | team = 6.0 |
| 2 | acap = 5.0 |
| 3 | pmat = 6.0 |
| 4 | resl = 6.0 |
| 5 | sced = 1.5 |
| 6 | prec = 6.0 |
| 7 | site = 6.0 |
| 8 | pcon = 5.0 |
| 9 | flex = 6.0 |
| 10 | apex = 5.0 |
| 11 | plex = 5.0 |
| 12 | sced = 1.0 |
| 13 | pcap = 5.0 |
| 14 | ltex = 5.0 |
| 15 | peer = 6.0 |
| 16 | rely = 5.0 |

# References: STAR/NOVA


- Understanding the Value of Software Engineering Technologies Phillip Green II, Tim Menzies, Steven Williams, Oussama El-Rawas, ASE 09 Proceedings of the fourth IEEE/ACM international conference on Automated software engineering (to appear). Available from tim@menzies.us
- Andres S. Orrego, Tim Menzies, Oussama El-Rawas: On the Relative Merits of Software Reuse. ICSP 2009: 186-197
- "How to Avoid Drastic Software Process Change (using Stochastic Statbility)" by T. Menzies and S. Williams and O. El-waras and B. Boehm and J. Hihn. ICSE09 . Available from http://menzies.us/pdf/08drastic.pdf .
- "Accurate Estimates without Calibration" by T. Menzies and O. Elrawas and B. Barry and R. Madachy and J. Hihn and D. Baker and K. Lum. International Conference on Software Process 2008 . Available from http://menzies.us/pdf/08icsp.pdf .
- "The Business Case for Automated Software Engineerng" by T. Menzies and O. Elrawas and J. Hihn and M. Feathear and B. Boehm and R. Madachy. ASE 07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering pages 303--312 location = Atlanta, Georgia, USA Available from http://menzies.us/pdf/07casease-v0.pdf
- "On the Value of Stochastic Abduction (if you fix everything, you lose fixes for everything else)" by T. Menzies and O. Elrawas and D. Baker and J. Hihn and K. Lum. International Workshop on Living with Uncertainty (an ASE07 co-located event) 2007 . Available from http://menzies.us/pdf/07fix.pdf .

## “2. Hard to explain all that random thrash?”

NASA AMES



WVU

TAR3

(with Karen Gundy-Burlet and Greg Gay):

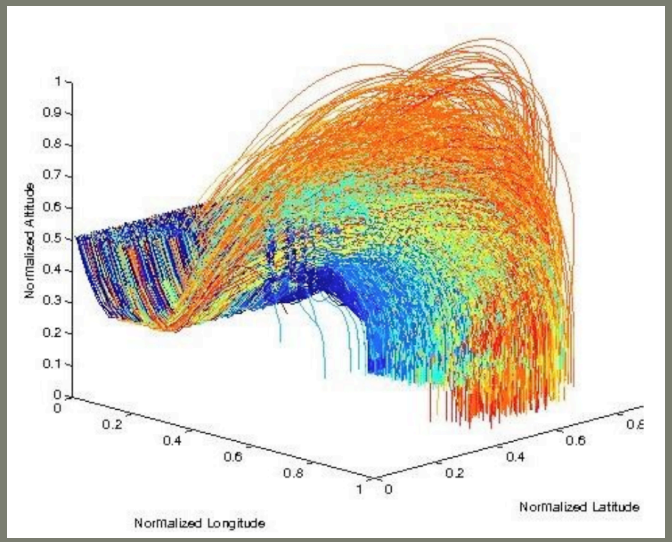
[IEEE Computer03, AIReview07]:

stochastic rule generation

Applications at NASA

# Understanding NASA's simulators

- "skip re-entry"
  - A quick dip (and out) into the upper atmosphere: burns off speed
  - GNC: Guidance navigation and control (300+ parameters)
  - Tiny variations in apogee of the "skip" = large errors at landing
- GNC software complex
  - No analytical solutions.
  - Must be simulated
  - Overdose of information

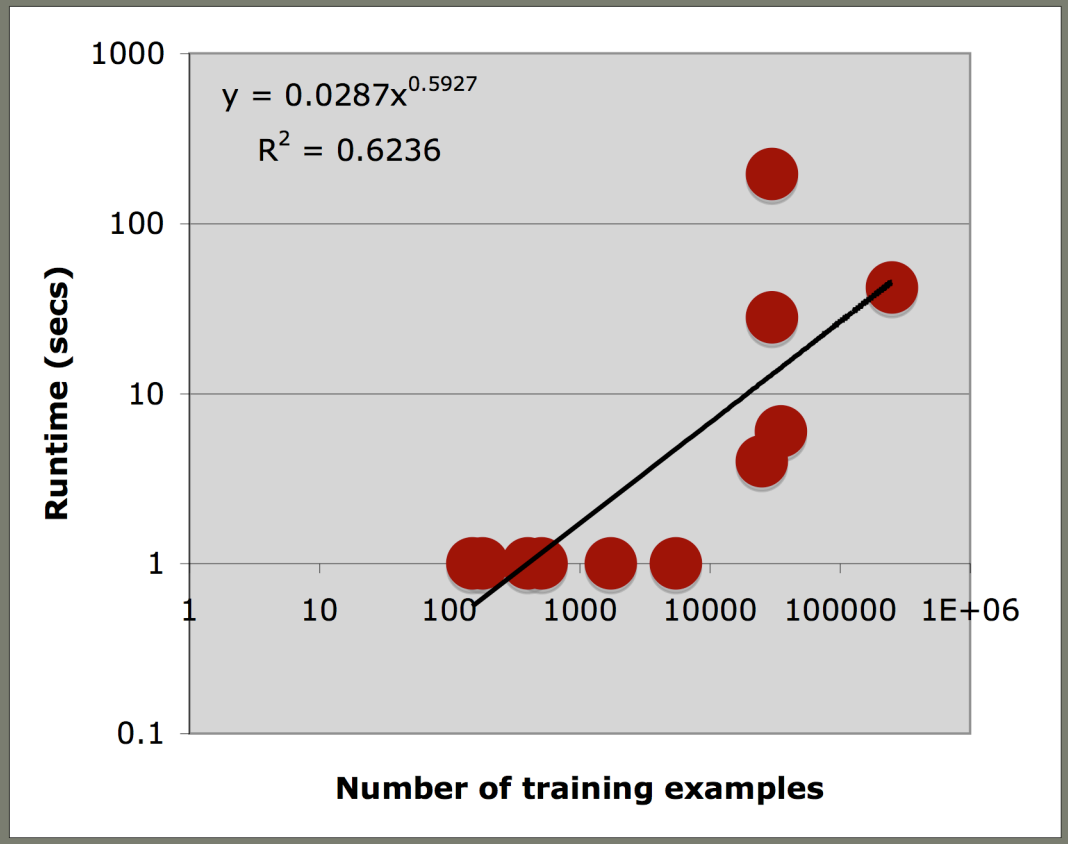



# A stochastic trick

- Assume “master variables” [Crawford & Baker’94]
  - A few variables that set the rest
  - Only those variables matter
- No need to hunt for the masters,
  - They’ll find you
- If master variables exist, they must be used in each simulation
  - By definition
- Add an oracle to the output
  - e.g. distance to touch down
  - Scores “good” / “bad”
- If master variables
  - Tiny explanations
  - Only a few rule small rules

# The TAR3 "treatment" learner

- Sort all attr=range
  - by "good" / "bad"
- Lives = 5
- While (Lives--) > 0
  - 100 times repeat
    - Build one rule
      - picking N ranges,
      - Favoring ranges higher in the sort
  - Add top 20 rules to "best"
  - If best has grown, then Lives++
- Return best

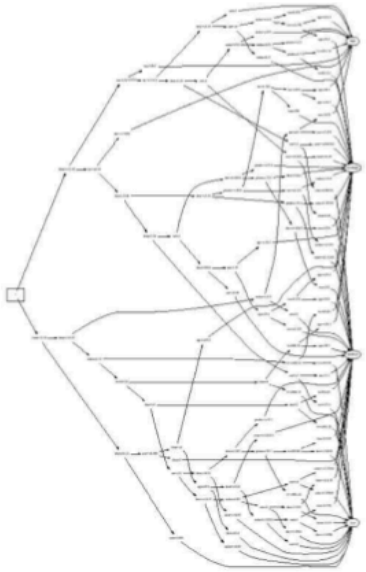


# Results (on UCI data)

- Not a “classifier”
  - But a “minimal contrast set learner”
- Not “what is”
  - But “what to do”

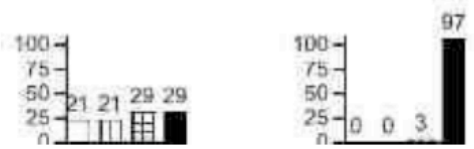
Decision tree classifiers (what is)

find good housing in Boston



A description of all things is much larger than …

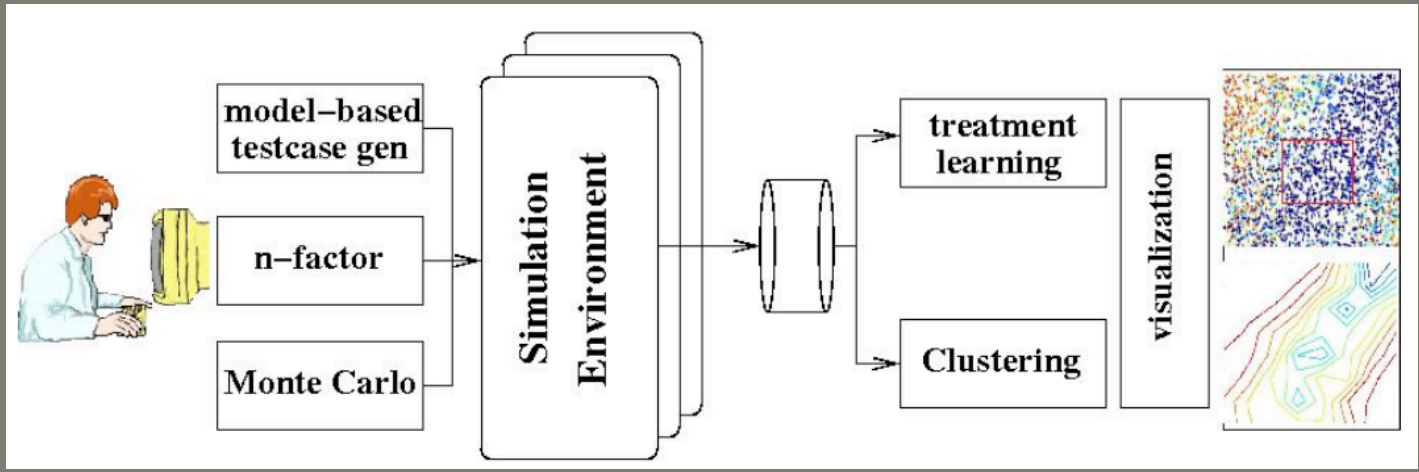
Treatment learning (what to do)




$6.7 \leq RM < 9.8 \wedge$
$12.6 \leq PTRATION < 15.9$

… a description of the essential differences between things

# Results (on NASA GNC data)




- From 300 parameters
  - Found 4 ranges that divide output space
  - MAS cgy location wet=0.0056. . . 0.0079
  - MAS lyz wet=-0.96 . . . -0.834
  - MAS wet=68.87. . . 69.49
  - INI rotx=0.929. . . 3.00
- Only need to view six 2-D graphs

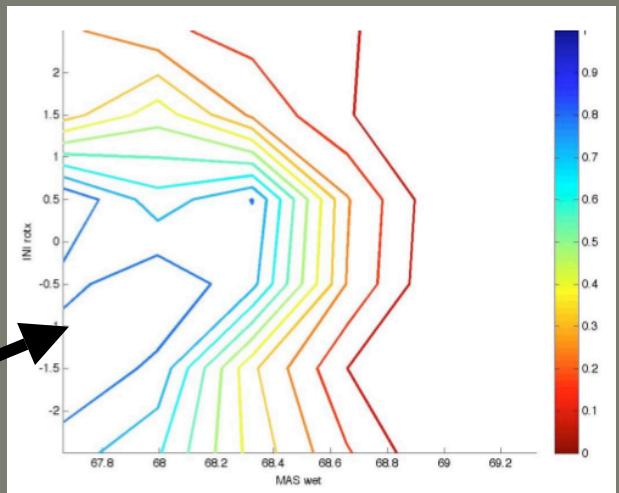



**Figure 7.** Likelihood of successful flight relative to initial x rotation and mass

# Discussion

- Despite 4 years of effort:
  - NASA has found nothing faster/better than TAR3
    - To find the sweet spots in their simulations.
    - And believe me, they've tried
- Implications for models:
  - Maybe "master variables" are very common
  - Would explain the curious success of stochastic analysis
- Implications for explanation (and control):
  - If an effect is common, and stand-out
  - Random probing will quickly stumble across it
- Q: "Hard to explain all that random thrash?"
  - A: random thrash simplifies understanding / explaining
  - Even for complex simulations.

- BTW, we have some new tricks….

# References: TAR3


- "Parametric Analysis of a Hover Test Vehicle Using Advanced Test Generation and Data Analysis" by K. Gundy-Burlet and J. Schumann and T. Menzies and T. Barrett. AIAA Aerospace, 2009.
- Johann Schumann, Karen Gundy-Burlet, Corina S. Pasareanu, Tim Menzies, Tony Barrett: Tool Support for Parametric Analysis of Large Software Simulation Systems. ASE 2008: 497-498
- "Application of a broad-spectrum quantitative requirements model to early-lifecycle decision making" by M. Feather and S. Cornford and K Hicks and J. Kiper and T. Menzies. IEEE Software May 2008 . Available from http://menzies.us/pdf/08ddp.pdf .
- "Parametric Analysis of ANTARES Re-entry Guidance Algorithms Using Advanced Test Generation and DAta Analysis" by K. Gundy-Burlet and J. Schumann and T. Menzies and T. Barrett. 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space 2008 . Available from http://menzies.us/pdf/08antares.pdf .
- "Just Enough Learning (of Association Rules): The TAR2 TreatmentLearner" by T. Menzies and Y. Hu. Artificial Intelligence Review 2007 Available from http://menzies.us/pdf/07tar2.pdf .
- "LEAN = (LURCH+TAR3) = Reusable Modeling Tools" by T. Burkleaux and T. Menzies and D. Owen. Proceedings of WITSE 2005 . Available from http://menzies.us/pdf/04lean.pdf .
- "Data Mining for Very Busy People" by T. Menzies and Y. Hu. IEEE Computer November 2003 . Available from http://menzies.us/pdf/03tar2.pdf .
- "Simulations for very early lifecycle quality evaluations" by E. Chiang and T. Menzies. Software Process: Improvement and Practice pages 141-159 2003 . Available from http://menzies.us/pdf/03spip.pdf
- "Optimizing Spacecraft Design Optimization Engine Development: Progress and Plans" by S. L. Cornford and M. S. Feather and J.R. Dunphy and J. Salcedo and T. Menzies. Proceedings of the IEEE Aerospace Conference, Big Sky, Montana 2003 . Available from http://menzies.us/pdf/03aero.pdf .
- "Model-Based Software Testing via Treatment Learning" by D. Geletko and T. Menzies. IEEE NASE SEW 2003 . Available from http://menzies.us/pdf/03radar.pdf .
- "Converging on the Optimal Attainment of Requirements" by M.S. Feather and T. Menzies. IEEE Joint Conference On Requirements Engineering ICRE02 and RE02, 9-13th September, University of Essen, Germany 2002 . Available from http://menzies.us/pdf/02re02.pdf .

# “3. Considered reflection is better?”

U. Western Ontario

NIGHTHAWK

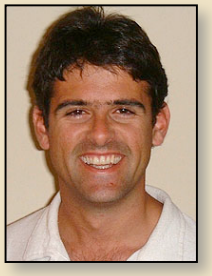
(with Jamie Andrews)

[ASE07, PROM09, TSE10?]

optimizing test case generation

# The Treaty of Camden Lock

- A stroll one sunny evening, England, 2007



Willem Visser:
Test case generation via considered reflection over the internal details of the software (symbolic execution)

Tim Menzies:
Thrash the input space, Don't worry about the internal details

RESOLVED: initial random probes yields much information, quickly, cheaply.

UNRESOLVED: assuming subsequent considered reflection, who does better?

# NIGHTHAWK: Stochastic unit test generation

- Unit tests = a sequence of functions…
- Value pools = variables set by $F_i$, used by $F_{j>I}$
- NIGHTHAWK: genetic algorithm that controls
  - Number of functions in a test (N)
  - Value pooling

a:= F1(); b:=F2(a); c:=F3(a,b); ... $F_N$

| SLOC | Source file |
|---|---|
| 9 | LHashSet |
| 17 | Stack |
| 46 | HashSet |
| 62 | TreeSet |
| 103 | LHashMap |
| 150 | ArrayList |
| 200 | Vector |
| 203 | PQueue |
| 227 | LinkedList |
| 239 | EnumMap |
| 355 | Hashtable |
| 360 | HashMap |
| 384 | WHashMap |
| 392 | IHashMap |
| 562 | TreeMap |
| 2492 | Properties |

**Figure 1a**: Case studies

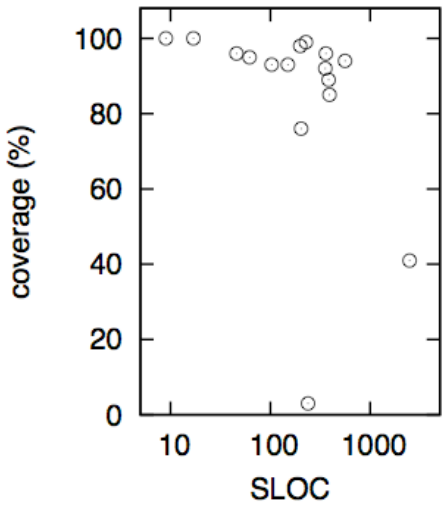



**Figure 1b**: SLOC vs line coverage. Median line coverage= 93%. Worst line coverage=3% from EnumMap.

Median line coverage = 93%

- More than other GA test generators
- NIGHTHAWK is a wrapper, no need for compilers on the source code
- Similar coverage to that seen with symbolic execution methods (sorry Willem)
- Scales to larger systems

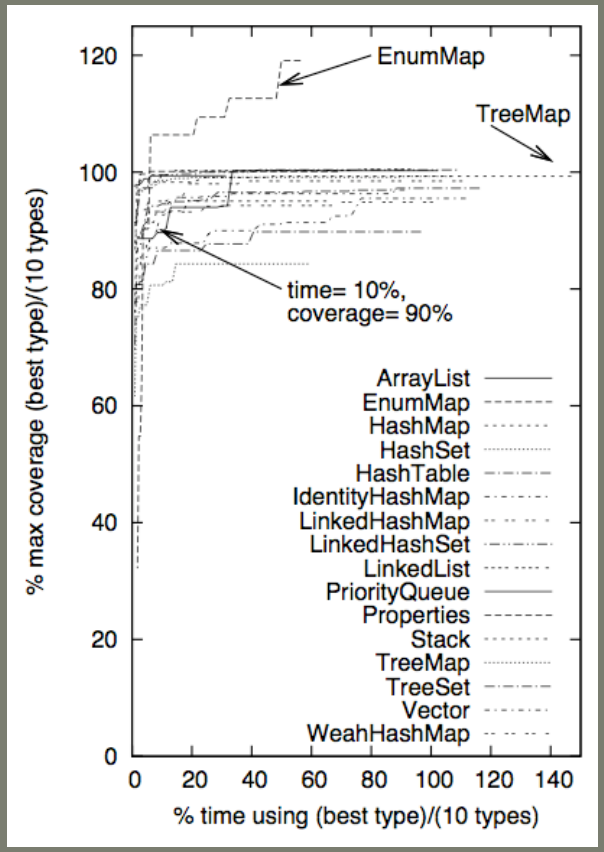
# Optimizing NIGHTHAWK with Stochastic Search

- NIGHTHAWK's GA mutates 10 operators:

| Rank | Gene type $t$ | $avgMerit$ |
|---|---|---|
| 1 | numberOfCalls | 85 |
| 2 | valuePoolActivityBitSet | 83 |
| 3 | upperBound | 64 |
| 4 | chanceOfTrue | 50 |
| 5 | methodWeight | 50 |
| 6 | numberOfValuePools | 49 |
| 7 | lowerBound | 44 |
| 8 | chanceOfNull | 40 |
| 9 | numberOfValues | 40 |
| 10 | candidateBitSet | 34 |

RELIEF feature subset selector Kira & Rendell'92, Kononenko'94

1. Take every <mutant,coverage>
   - Mutant : {attr1=val1, attr2=val2,…}
2. Discretize coverage into {lo,med,hi}
3. 250 times repeat:
   - Randomly pick one mutant of class C
   - Hit : closest neighbor with class=C
   - Miss : closest neighbor with class != C
   - Score each attribute by their delta in Hit and Miss




The top-ranked mutator found by RELIEF achieves 90% of the best coverage in 10% of the time

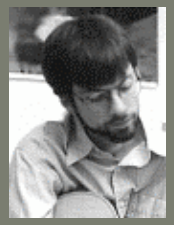
# Discussion

- Implications for Gas
  - GA design something of a black-art
  - Use stochastic analysis to prune irrelevant decisions
- Implications for modeling
  - Complex devices can be audited using stochastic analysis
  - Don’t prune prematurely: try everything
    - Humans propose
    - Stochastic dispose
- Implications for stochastic analysis
  - Useful to augment “random” with domain biases
    - If you’ve got domain knowledge, use it
    - E.g. value pools

# References: NIGHTHAWK


- Controlling Randomized Unit Testing With Genetic Algorithms James H. Andrews, Member, Tim Menzies, and Felix C. H. Li . Submitted to IEEE TSE 2010. Available from tim@menzies.us.
- "On the Value of Combining Feature Subset Selection with Genetic Algorithms: Faster Learning of Coverage Models", PROMISE 2009, Jamie Andrews and Tim Menzies. Available from http://promisedata.org/pdf/2009/11_Andrews.pdf
- "Nighthawk: A Two-Level Genetic-Random Unit Test Data Generator" by J.H. Andrews and F.C.H. Li and T. Menzies. IEEE ASE07 . Available from http://menzies.us/pdf/07ase-nighthawk.pdf .

# "4. Nondeterminism is the enemy of reliability?"



Owen: Messiah College

Cukic: WVU

LURCH:
(with David Owen and Bojan Cukic
and Mats Heimdahl and Jimim Goa)
random search for software verification
[Owen07, DSN08, ISSRE08, TSE10?]

Goa: UMN

Heimdahl: UMN

# LURCH : Stochastic Exploration of State Machines

- Theory = <machine+>
- Machine = <transition*, state+>
- Transition = <now, in, out, next>
- State= <#X | _X | X. | X>
  - #X = progress cycle
  - _X = faulty state
  - X. = legal resting state
  - X = other
  - States mutually exclusive
  - One state is the “default state”
- Global state:
  - One state assignment to every machine

- Machine 1 + 2:
  - Two processes
  - competing for same resource.
- Machine 3:
  - Property model: are 1&2 both in critical sections simultaneously (a fault).

```
#define FALSE 0
#define TRUE 1

enum { A, B } turn = A;
int a = TRUE, b = TRUE;

void before() {
  turn = A;
  a = b = TRUE;
}

%%

a1;  -;          {a = TRUE;};  a2;
a2;  -;          {turn = B;};  a3;
a3;  (!b);        -;           acs;
a3;  (turn == A); -;           acs;
acs; -;          {a = FALSE;}; a5.

b1;  -;          {b = TRUE;};  b2;
b2;  -;          {turn = A;};  b3;
b3;  (!a);        -;           bcs;
b3;  (turn == B); -;           bcs;
bcs; -;          {b = FALSE;}; b5.

ok;  acs,bcs; -; _fault.
```

# LURCH example: Mutual Exclusion (2)

- An error is introduced by deleting the input condition for the transition marked in red:

```
a1;  -;          {a = TRUE;};  a2;
a2;  -;          {turn = B;};  a3;
a3;  (!b);        -;            acs;
a3;  (turn == A); -;            acs;
acs; -;          {a = FALSE;}; a5.
```

Machine A no longer checks whether B has claimed the shared resource before entering its critical section.

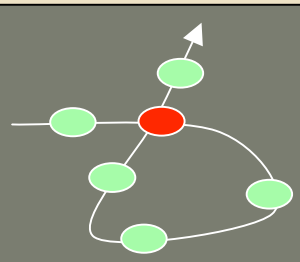
```
a1;  -;          {a = TRUE;};  a2;
a2;  -;          {turn = B;};  a3;
a3;  -;          -;            acs;
a3;  (turn == A); -;            acs;
acs; -;          {a = FALSE;}; a5.
```

# Inside LURCH

- Inputs=initializations
- While not stop do
  - time++
  - for m ∈ machines do
    - for t ∈ m.transitions do
      - if t.now == m.now and t.in
      - then begin
        - » ok(t.out)
        - » m=t.next and t.out
        - » skip to next machine
  - Inputs = global state
  - globalStatePath[time]=global state
- done

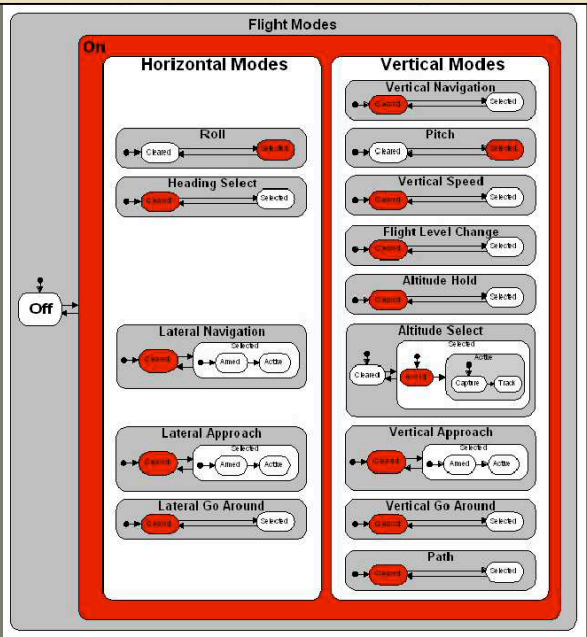
- On "saturation"
  - each global state hashed to a 32-bit unsigned int
  - < X% new global states seen in this repeat
  - default = 0.01%
- Too many "collisions"
  - same global state reached over all different repeats
  - requires inter-iteration memory
  - default=250

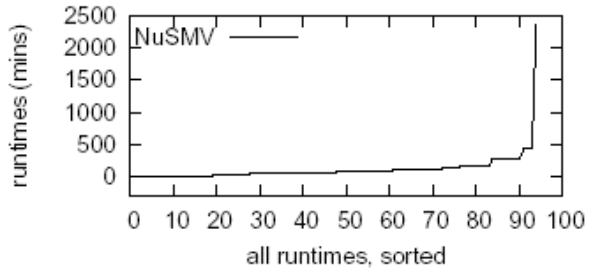
- Search in random order
- Use one transition/machine
- Block if t.out inconsistent



# LURCH is fast(er)

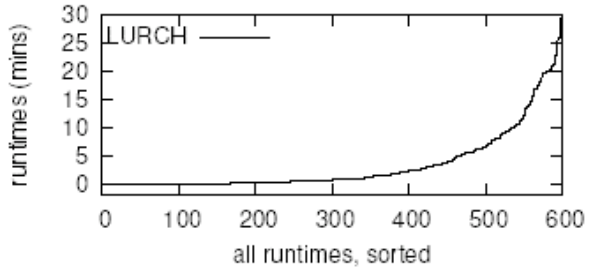
- Flight Guidance System:
  - Rockwell Collins Inc.
  - Flight Control System
  - Pitch and Roll Commands
  - Complex mode logic
- Method:
  - Convert  to LURCH, NuSMV
  - Seed errors in using historical records




| | median | mean | max |
|---|---|---|---|
| NuSMV | 73 | 125 | 2358 |

**Figure 1. Time (rounded to minutes) to find violations using NuSMV.**




| | median | mean | max |
|---|---|---|---|
| LURCH | 1 | 3 | 29 |

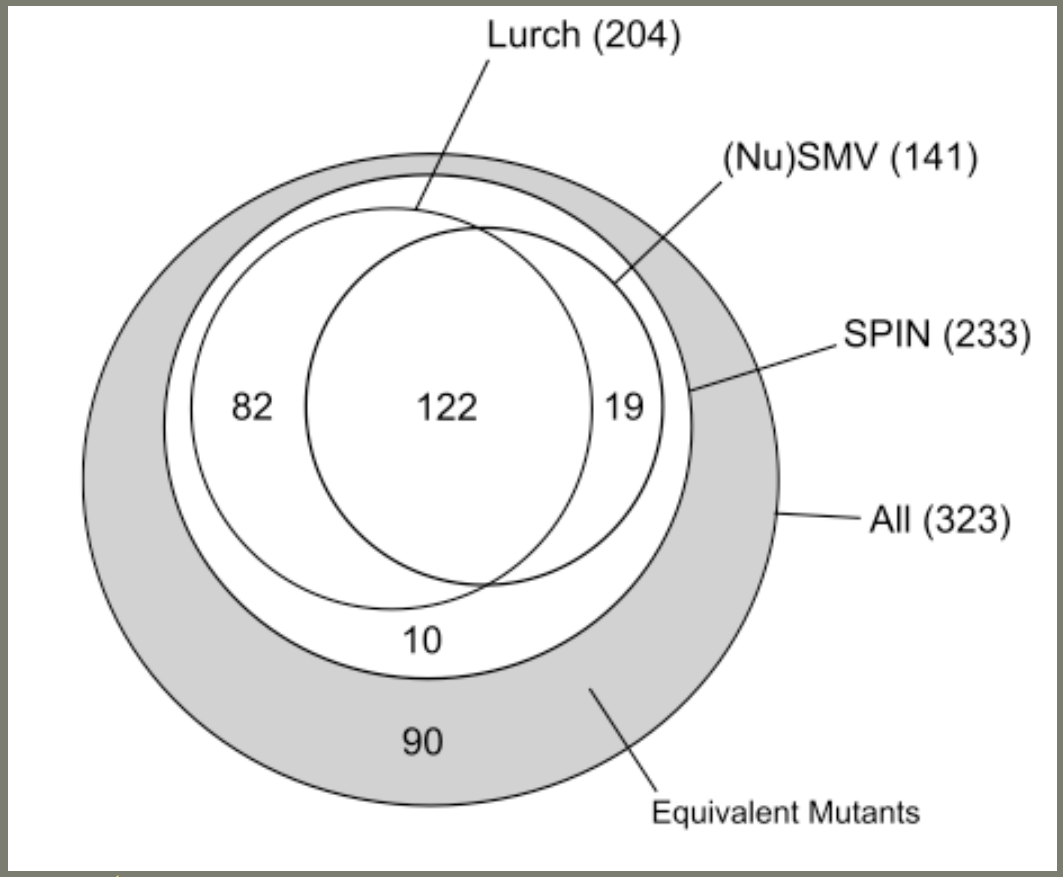
**Figure 2. Time (rounded to minutes) to find violations using LURCH.**

# LURCH is incomplete (a little)

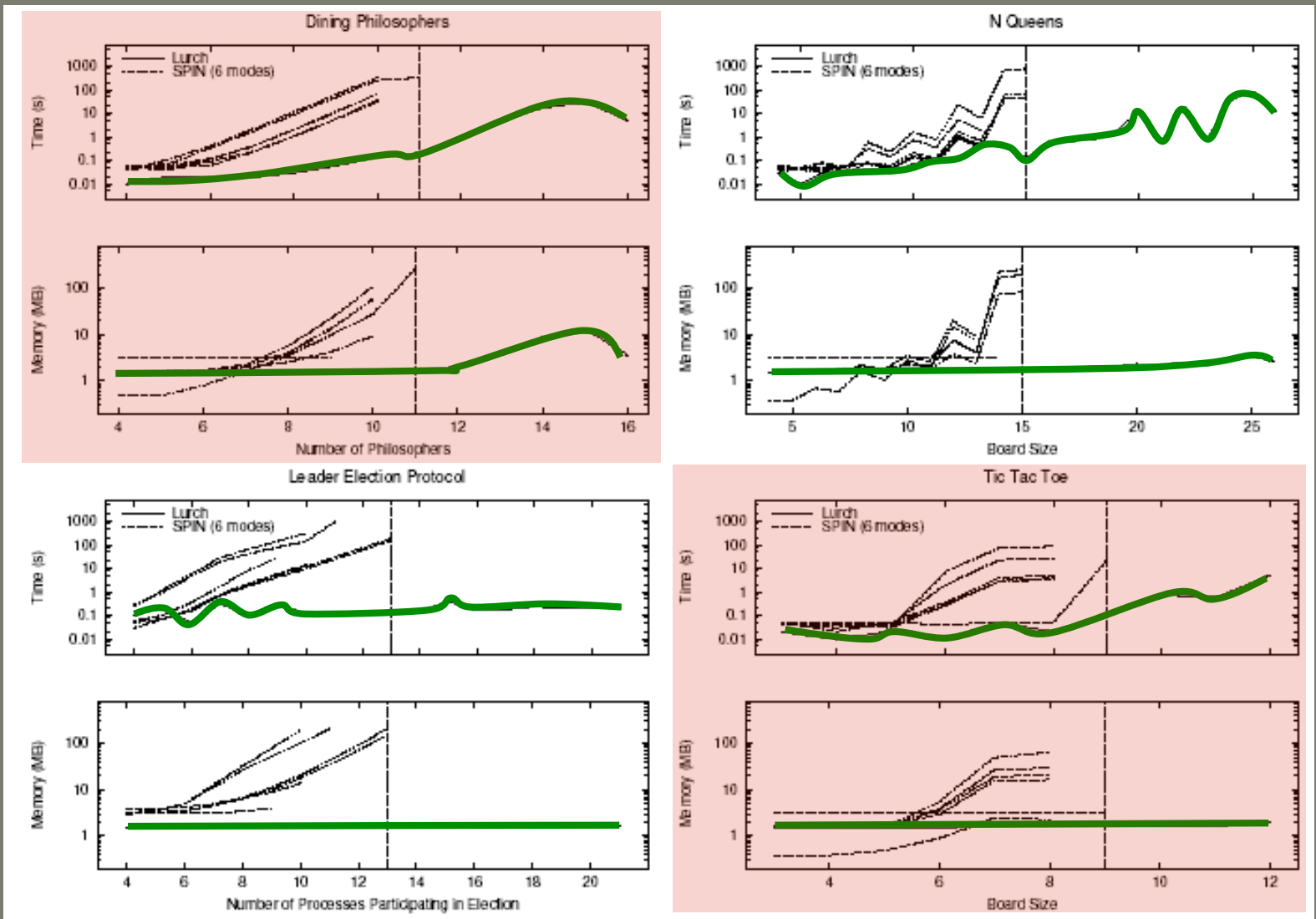
- Personnel Access Control system for NSA
  - Ported to SPIN, NuSMV, LURCH

| Label | Description | Example |
|---|---|---|
| AOR | Arithmetic Operator Repl. | `+` → `-` |
| CRP | Constant Repl. | `1` → `2` |
| EVR | Enumerated Type Val. Repl. | `a` → `b` (where `a` and `b` are possible values for enumerated type) |
| IOR | Implication Operator Repl. | `=>` → `<=>` |
| LCR | Logical Connector Repl. | `AND` → `OR` |
| ROR | Relational Operator Repl. | `<` → `<=` |
| SND | SCR *Next* Operator Del. | `'` → |
| SOR | SCR Event Operator Repl. | `@T` → `@F` |
| UOD | Unary Operator Del. | `NOT` → |
| VRP | Variable (same type) Repl. | `x` → `y` (where `x` and `y` are variables of same type) |

323 mutations

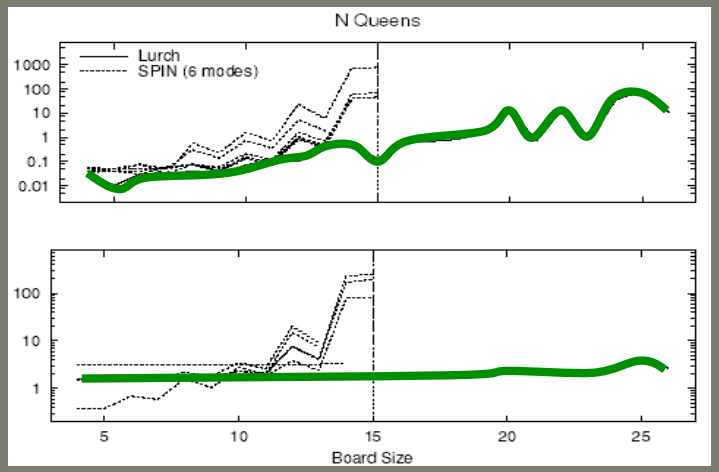


# LURCH works on models too big for more thorough approaches

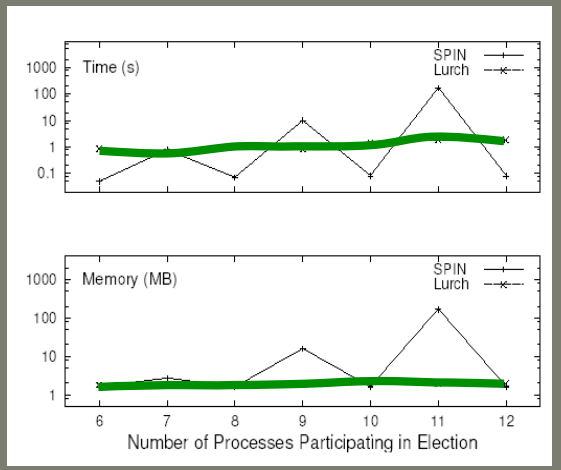


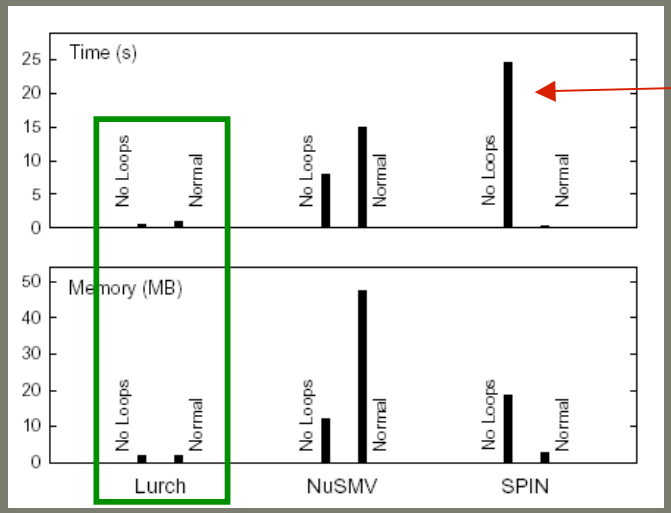
# LURCH is stable

N-queens




Holzmann's leader election model (with errors Injected)




Got to find the shortest path: (hard for SPIN)

Holzmann certified Promela

Dining philosophers:
1) Normal= looping
2) No loops (never get hungry again)

# Harder to break a stochastic algorithm

- A “critic” of a deterministic algorithm:
  - Finds some input order that generates worst-case behavior
  - E.g. worst case Insertion Sort:
    - input array sorted in decreasing order.
- Randomized algorithm:
  - Samples the space of possible algorithms
  - Hard (++) to find the input order that defeats the randomized search

# Discussion

- For small models, LURCH may miss some things
  - But still finds hundreds of errors
- Not distracted by tiny changes to the models
  - Stability results
- For larger models, it may be the only tool that can explore the software's state space
  - Memory and runtime results
- So, stochastic search increases reliability of larger systems

# References: LURCH


- "Combining Diverse Translation and Verification  Tools to Detec Faults" Tim Menzies with David Owen, Bojan Cukic, Member,  Submitted to IEEE TSE, 2010. Available from tim@menzies.us.
- "On the Distribution of Property Violations in Formal Models: An Initial Study" by J. Gao and M. Heimdahl and D. Owen and T. Menzies COMPSAC 06 2006 . Available from http://menzies.us/pdf/06compsac.pdf
- "Lurch: a Lightweight Alternative to Model Checking" by D. Owen and T. Menzies. SEKE 03 2003 . Available from http://menzies.us/pdf/03lurch.pdf  .
- "On the Advantages of Approximate vs. Complete Verification: Bigger Models, Faster, Less Memory Usually Accurate" by David Owen and Tim Menzies and Mats Heimdahl and Jimin Gao. IEEE NASE SEW 2003 . Available from http://menzies.us/pdf/03lurchc.pdf  .

# Conclusion

# Once again: What was the problem / solution?

- 21st century software problems characterized by
  - larger assemblies
  - Less knowledge of the parts
- Can we reason about such system?
  - Is anything certain amidst all the possibilities?
  - Can such reasoning scale to large systems?

Seek islands of certainty within an ocean of possibilities

# Advocacy via case studies

- Much prior research has advocated stochastic analysis
  - Bayesian
  - Long-tailed distributions in backtracking
  - phase transition effects,
  - log-normal distribution of reachable states
  - Keys/ collars/ clumps
  - Etc
- To many audiences, such arguments fall on deaf ears
  - So this talk has offered practical examples of utility of stochastic analysis for SE

1. "No value added by random thrashing"
   - NOVA / STAR enables process option ranking, without local tuning
2. "Hard to explain all that random thrash"
   - TAR3 generates tiny explanations of complex models
3. "Considered reflection is better"
   - GAs + value pools perform as well as symbolic execution, simpler to use.
   - Also, can use stochastics to improve GA's design
4. "Non-determinism is the enemy of reliability"
   - LURCH can check properties in models that defeat other methods

# Finally…..

- "Random" is not "crazy"
  - It's a random world.
  - Accept it.
  - Exploit it

- While the exact methods differed…
  - … the general approach was constant


- Advice:
  - Code the stochastic version
    - before the more complex deterministic version
  - At the very least,
    - you get a baseline result
      - rapidly, cheaply
  - At the very most,
    - you get much, much more

# Questions, comments?

# Back-up slides

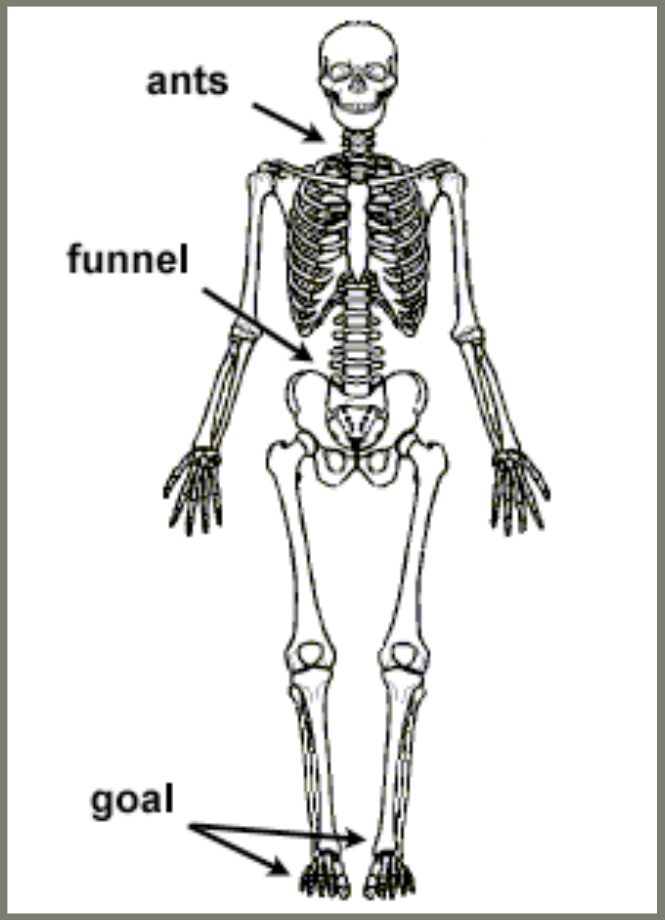
# TAR3 pseudo-code

- Assume clumps and collars
  - Just thrash around some.
- Build treatments
  $\{R_1 \wedge R_2 \ldots \wedge R_X\}$ of size $X$
  - FIRST try $X = 1$
  - THEN use the $X = 1$ results to guide the $X > 1$ search.
- Hu [2002] :: grow *treatments* via a stochastic search.
  - Discretization: equal frequency binning
- Empirically:
  - Run times linear on treatment SIZE, number of examples
  - Works as well as TAR2's complete search

```
function ONE(x = random(SIZE) )
   x timesDo
      treatment = treatment + ANYTHING()
   return treatment

function ANYTHING( )
   return a random range from CDF(lift1)

function SOME( )
   REPEATS timesDo
      treatments = treatments + ONE()
   sort treatments on lift
   return ENOUGH top items

function TAR3(lives = LIVES )
   for every range r do lift1[r]= lift(r)
   repeat
     before = size(temp)
     temp   = union(temp, SOME())
     if (before==size(temp))
     then lives--
     else lives = LIVES
   until lives == 0
   sort temp on lift;
   return ENOUGH top items
```

Useful defaults: <SIZE=10, REPEATS=100, ENOUGH=20, LIVES=5>

# Random search through a space with "funnels"




- Models with many variables
  - Funnel= just the few that count
- Don't have to search for funnels
  - They'll find you
- Randomly selected paths must pass through the funnel
  - by definition

# Funnel theory: not so crazy?

**6 scheduling problems**
**[Crawford & Baker 94]**

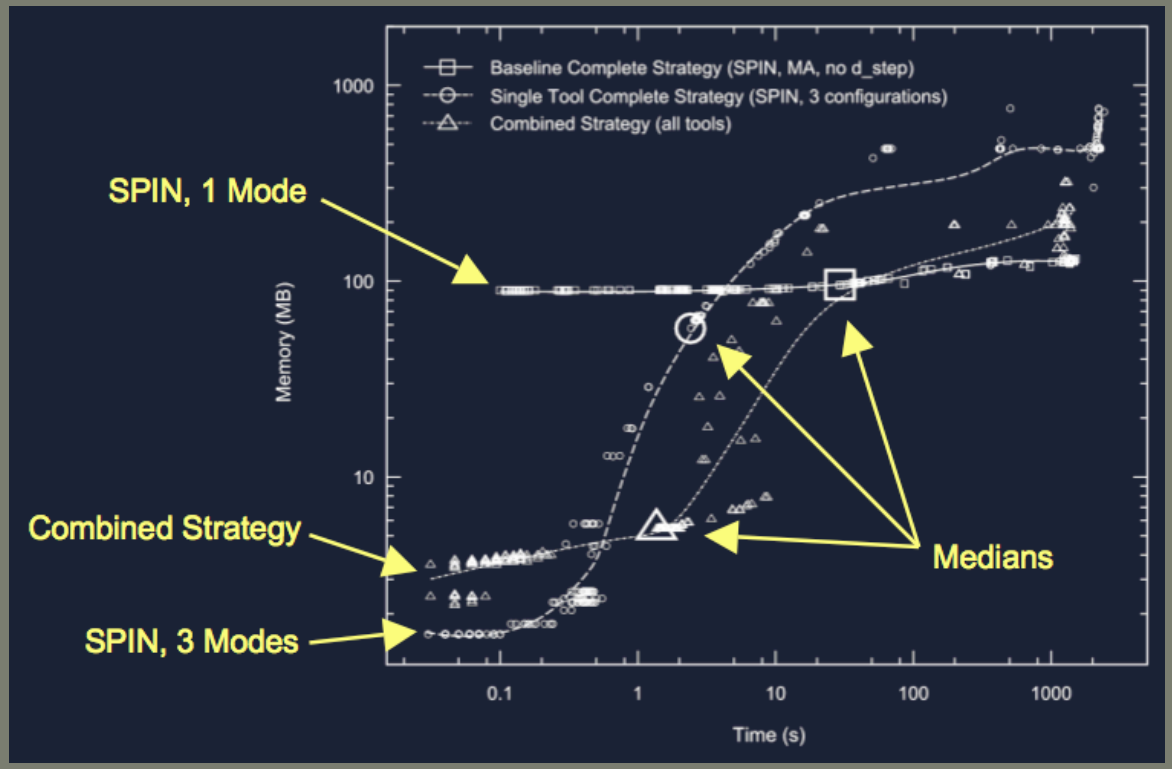
```
for TRIES := 1 to MAX-TRIES
  {set all vars to unassigned;
  loop { if      everything assigned
         then return assignments;
         else  pick any var v at random;
               set v's value at random;
               forwardChainFrom(v);
               if     contradiction
               then exit loop;
               fi
         fi
  }}
return failure
```

| | TABLEAU | | ISAMP | | |
|---|---|---|---|---|---|
| | % | secs | % | secs | retries |
| A | 90 | 225.4 | 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 | 81.6 | 100 | 68 | 252 |

**Theory:**
- **master-variables**
- **set all the other “slave” variables**
- **a.k.a. funnels**

# Greater speed via "combinations"

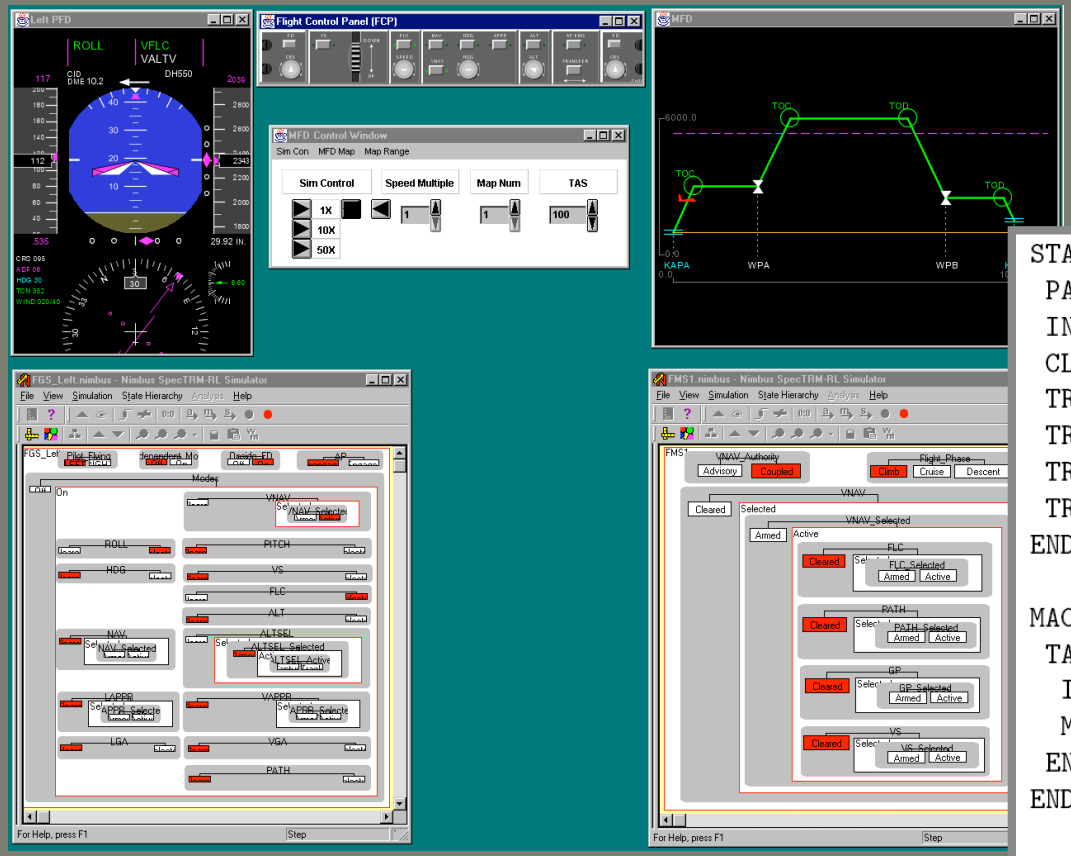
- Take N verification tools: Spin, NuSMV, LURCH, Salsa, etc
- Sort them by expected runtimes
- The "combined strategy" : only next tool if last tool found no errors
- Experiments with the Personnel Access Control system for NSA:




Fast retest very useful during maintenance (e.g. during refactoring)

Maturing the dialogue. Not which tool is "best" but how to lever all the tools that are out there.

# Models for quality

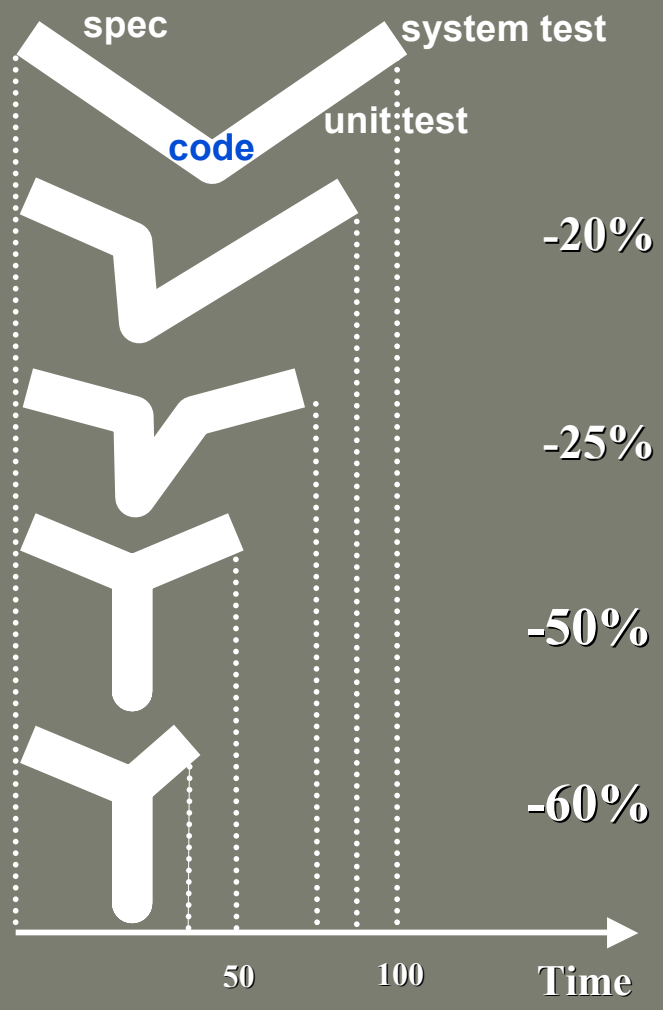



## Flight guidance systems (Rockwell-Collins)

```
STATE_VARIABLE ROLL : Base_State
 PARENT           : Modes.On
 INITIAL_VALUE    : UNDEFINED
 CLASSIFICATION   : State
 TRANSITION UNDEFINED TO Cleared IF NOT Select_ROLL()
 TRANSITION UNDEFINED TO Selected IF Select_ROLL()
 TRANSITION Cleared TO Selected IF Select_ROLL()
 TRANSITION Selected TO Cleared IF Deselect_ROLL()
END STATE_VARIABLE

MACRO Select_ROLL() :
 TABLE
  Is_No_Nonbasic_Lateral_Mode_Active()     : T;
  Modes = On                               : T;
 END TABLE
END MACRO

MACRO Deselect_ROLL() :
 TABLE
  When_Nonbasic_Lateral_Mode_Activated()   : T *;
  When(Modes = Off)                        : * T;
 END TABLE
END MACRO
```

# From V to Y

- Manual coding
- Use of a “regular” automatic code generator
- Use of the qualifiable code generator as a verification tool
- Use of the qualifiable code generator as a development tool
- Use of proof technology



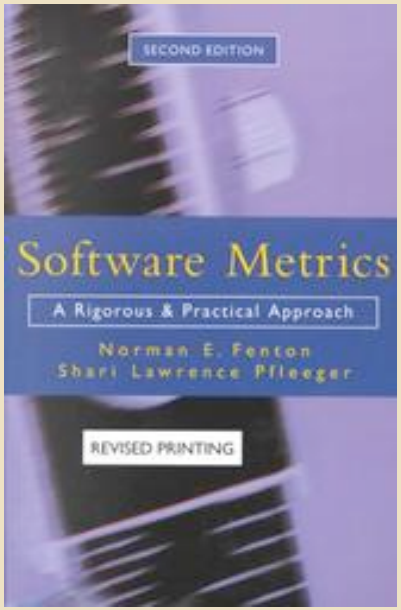
# How complex are our models?

- COLLARS- A small number few variables controls the rest:
  - DeKleer [1986]: minimal environments in the ATMS;
  - Menzies and Singh [2003]: tiny minimal environments
  - Crawford and Baker [1994]: master variables in scheduling;
  - Williams et al. [2003]: backdoors in satisfiability.
- CLUMPS
  - Druzdzel [1994]. Commonly, a few states; very rarely, most states;
  - Pelanek [2004]. straight jackets in formal models: state spaces usually sparse, small diameter, many diamonds.

25,000 states in IEEE 1394

# Problem$^2$

- As we push back the boundaries…
  - we enter uncharted territory

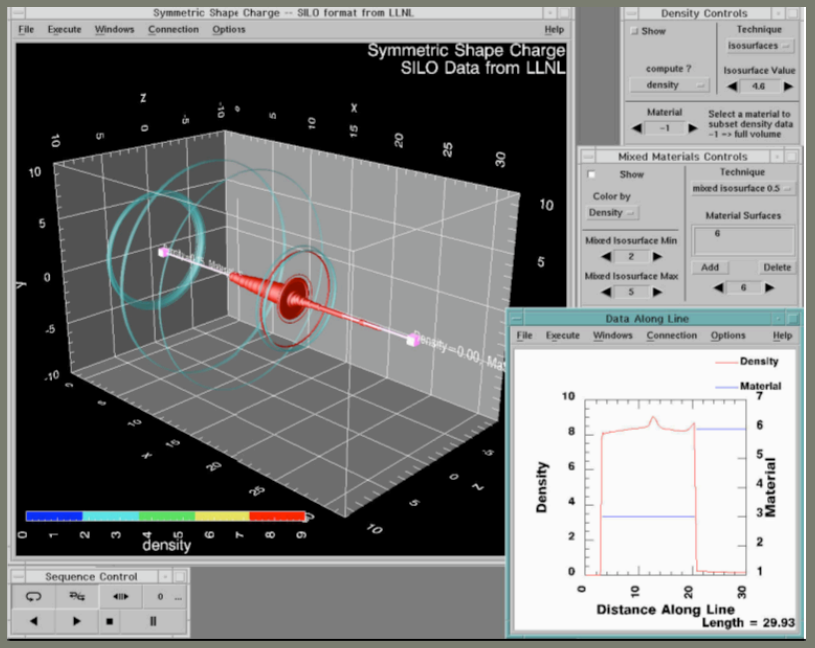



Fenton07: "….much of the current software metrics research is inherently irrelevant to the industrial mix … any software metrics program that depends on some extensive metrics collection is doomed to failure. "

e.g. After 26 years, Boehm collected less than 200 sample projects for the COCOMO effort database

# Monte Carlo + Visualizations

Conventional Monte Carlo followed by visualizations limited to less than 10-D.
e.g. how may dimensions are shown here?



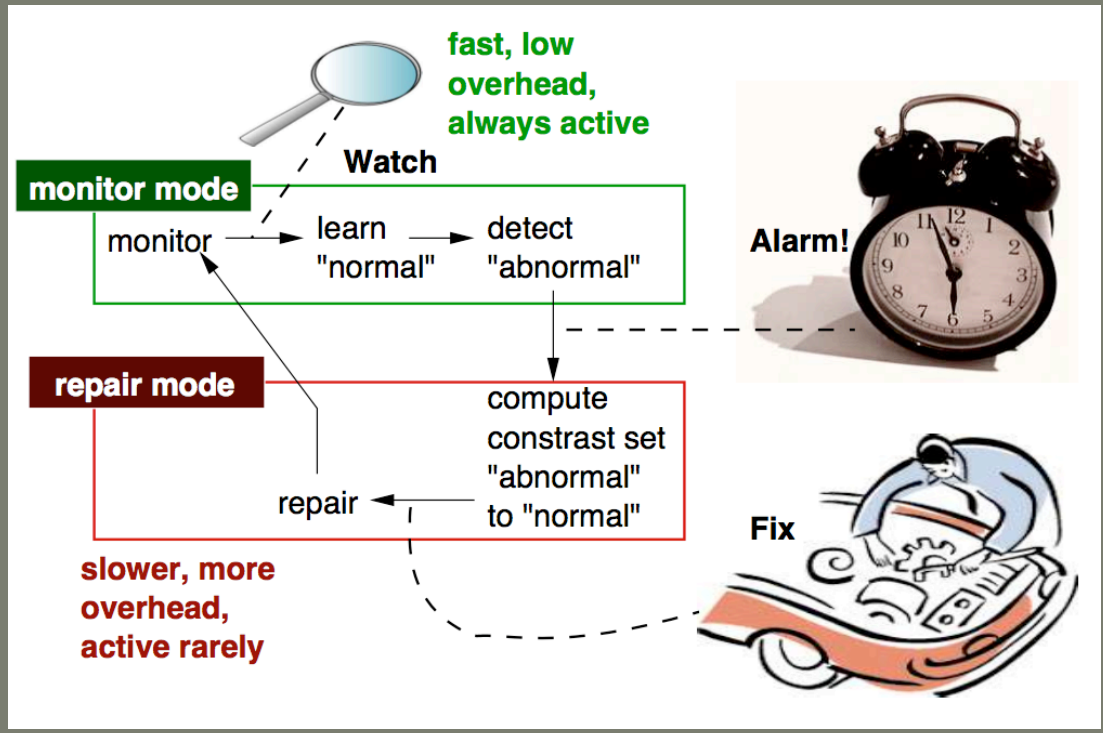
# Abduction (warning: very geeky slide)

- Abduction:
  - Theory + Assumptions ==> Goals
  - Not(Theory + Assumptions ==> Errors)
- If multiple possible assumption sets
  - Separate into “worlds”
  - Maximal w.r.t. size:
    contains no other worlds
  - Internally consistent:
    intra-world reasoning is fast
- What is the “best” world?
  - If cheapest traversal then
    abduction = planning
  - uses familiar concepts?
    abduction = explanation
- Cache worlds, monitor environment,
  - Delete any that clash with observations.
    Remaining worlds are remaining plans
  - Abduction = monitoring
- Worlds are sub-optimal?
  - Plan least cost path to better worlds
  - Abduction = tutoring

- What is wrong with this picture?
- Abduction is NP-hard
  - Generation of all worlds is slow
- Technical possibility
  - Don’t generate all
  - Sample some, at random
  - Stop generation when found “enough”
- LURCH= fast sampler
  - Wanted: incremental data mining

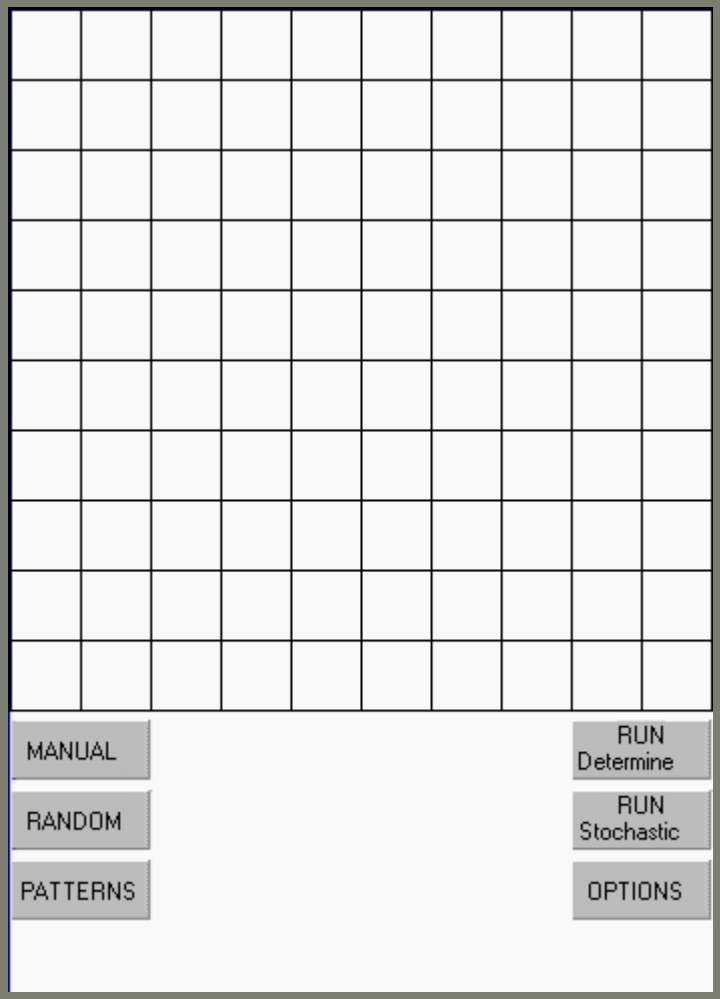
# LURCH = sub-routine for abduction: from “alert” to “repair”

- To verify, LURCH stumbles around the model
  - a lot
  - As a side=effect of all that peeking, can we learn something along the way?
- FP-trees*: used to detect frequent patterns in data steams
  - Faster than standard FP-trees
  - Instruments LURCH to find frequent one-item sets
- GENIC: incremental clusterer of frequent patterns
- SAVVY: a linear-time Bayesian contrast set learner (what is the difference between the patterns?)



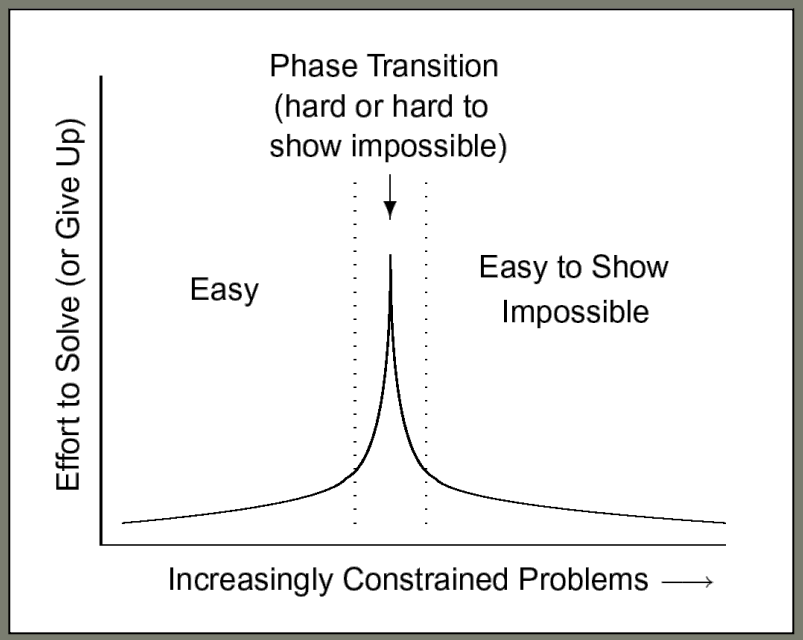

Now, finally, we arrive a “C” for control

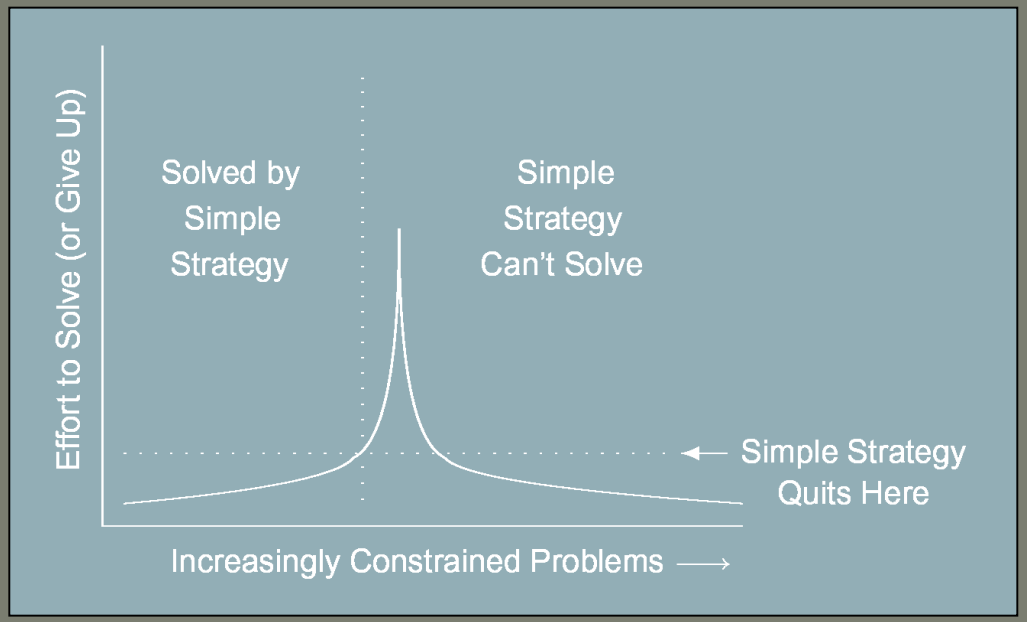
# Latin squares, again




**Animation by Bart Selman**

# The Phase Transition




**Backbone= variables shared by all solutions**
**Large backbone= long runtimes**
**Small backbone= short runtimes**