August 6, 2006
WP ref: http://unbox.org/all/trunk/doc/06/xomo2
Download this paper at http://unbox.org/all/trunk/doc/06/xomo2.pdf.

XOMO: Processing models within the RST test bed

Tim Menzies

Lane Department of Computer Science & Electrical Engineering, Morgantown, West Virginia University, USA
tim@menzies.us
URL: http://menzies.us

Abstract Previously, we have defined an iterative data mining method for learning better software product and process. That method was prototyped on part of the space shuttle abort software. Here, the same method is applied to artifacts in the Reliable Software Test bench under development at the AMES Research Center.

As before, our methods found ways to significantly reduce risks associated with a project. In particular, the mean estimated $\frac{defects}{KLOC}$ were reduced from 4 to 0.5. Further, the certainty in this estimate was vastly improved since the standard deviation on the estimated $\frac{defects}{KLOC}$ was reduced by two orders of magnitude from 3.4 to 0.05.

Better yet, our methods supported a fine-grained analysis of the merits of automated analysis vs execution-based testing. When such execution-based testing is impractical, we could identify what could be gained using (e.g.) just automatic analysis.

List of Figures

Contents

This research was conducted at West Virginia University under NASA sub-contract project 100005549, task 5g, award 1002193r. All software discussed here is available from http://unbox.org/wisp/xomo under the GNU Public License (version 2: see www.gnu.org/copyleft/gpl.html). Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

1 Introduction

The manager of any NASA project must answer three questions:

1. What level of risk is acceptable for this project?
2. How much risk does a project have?
3. How to reduce #2 to #1?

This paper is about questions two and three. Once the appropriate level of risk has been determined, then a project manager struggles to determine and reduce the project risk. Previously we have learned risk reduction strategies for space shuttle sub-system using three models:

- The COCOMO effort estimation model [2, p29-57];
- The COQUALMO defect model [2, p254-268];
- The THREAT schedule threat model [2, 284-291][1].

The models use dozens of variables, not all of which are known with certainty. For example, until software is completed, the exact size of a program may be unknown. Hence, exploring our effort, defect, and threat models requires exploring a space of possible model inputs.

To achieve this, our recommended approach uses Monte Carlo simulation and data mining in the cyclic manner of Figure 1:

- A Monte Carlo simulator repeatedly call some models, drawing model inputs from some pre-defined ranges.
- The output from the runs are then studied by data miners to find some "fitter" ranges to use in subsequent calls to the model.
- This cycle continues, defining narrower and narrower ranges; i.e. $ranges_{i+1} \subseteq ranges_i$. i

The cycle ends when new ranges are not "fitter" than old ones. Here, "fitter" is determined by some fitness function that scores model output: the fitter the input ranges, the higher the scores of the model output. For example:

- When processing COCOMO, COQUALMO, and THREAT, the fitness function would select for lower effort, lower defects, and fewer threats.
- For other models the tools et of Figure 1 remains mostly the same, but the fitness function changes (see Figure 2).

The tool set that implements Figure 1 using the COCOMO, COQUALMO, THREAT models and the TAR3 data miner [11] is called XOMO (pronounced "x-o-mow") [14]. XOMO uses TAR3 since this data miner assumes that busy managers of software projects don't need (or can't use) complex models. Rather, busy people need to know the *least* they need to do to achieve the *most* benefits. For example, when dealing with complex situations with many unknowns (e.g. developing software), it can be a wise tactic to focus your efforts on a small number of key factors rather than expending great effort trying to control all possibilities.

[1] Historical note: previously, we called "THREAT" the "Madachy schedule risk model" [14].

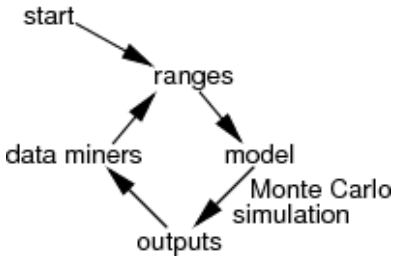

Fig. 1 Cyclic learning

This paper applies XOMO to a new model taken from the Reliable Software Test bench under development at the AMES Research Center. One item in RST is ARES, a model of the guidance and navigation control (GNC) system of ARES1 (formerly known as the CEV). Figure 1 found ways to significantly reduce issues within ARES. In particular, the mean estimated $\frac{defects}{KLOC}$ were reduced from 4 to 0.5.

Better yet, our methods supported a fine-grained analysis of the merits of automated analysis vs execution-based testing. When such execution-based testing is impractical, we could identify what could be gained using just (e.g.) automatic analysis.

Within NASA, Figure 1 has been applied to:

- Spacecraft design, where fitter means "covers more requirements and reduces most risk, costs the least" [5, 6].
- Software process control using:
  - A Chung-Mypolopous soft-goal graph where fitter means "cover more non-functional requirements" [4].
  - The COCOMO and THREAT models where fitter means "lower effort and fewer threats" [16];
  - The COCOMO and THREAT and COQUALMO models where fitter means "lower effort and fewer threats and lower defects" [14];
  - Qualitative inference diagrams where fitter means "higher quality" [15].
  - The IV&V "Silap" model that selects work break down structures for V&V where fitter means "lower risks" [7];

Outside of NASA, Figure 1 has been previously applied to:

- Circuit design, with fitter being "more bulbs shine" [10].
- Finite state machines, where fitter means "better chances of covering transitions" [17, 18].
- Economic policy where fitter means "longer human life" [8];
- Whiskey production, where fitter means "more alcohol" [3];
- Software process control using:
  - a CMM level 2 model, where fitter mean "chance of reaching a lost cost project" [12];
  - Discrete event simulation where fitter was defined by a utility function combining quality, expense, and development time [13].

Fig. 2 Prior applications of Figure 1.

	Definition	Low-end	Medium	High-end
Scale factors:				
flex	development flexibility	development process rigorously defined	some guidelines, which can be relaxed	only general goals defined
pmat	process maturity	CMM level 1	CMM level 3	CMM level 5
prec	precedentedness	we have never built this kind of software before	somewhat new	thoroughly familiar
resl	architecture or risk resolution	few interfaces defined or few risk eliminated	most interfaces defined or most risks eliminated	all interfaces defined or all risks eliminated
team	team cohesion	very difficult interactions	basically co-operative	seamless interactions
Effort multipliers				
acap	analyst capability	worst 15%	55%	best 10%
aexp	applications experience	2 months	1 year	6 years
cplx	product complexity	e.g. simple read/write statements	e.g. use of simple interface widgets	e.g. performance-critical embedded systems
data	database size (DB bytes/SLOC)	10	100	1000
docu	documentation	many life-cycle phases not documented		extensive reporting for each life-cycle phase
ltex	language and tool-set experience	2 months	1 year	6 years
pcap	programmer capability	worst 15%	55%	best 10%
pcon	personnel continuity (% turnover per year)	48%	12%	3%
plex	platform experience	2 months	1 year	6 years
pvol	platform volatility ($\frac{frequency\ of\ major\ changes}{frequency\ of\ minor\ changes}$)	$\frac{12\ months}{1\ month}$	$\frac{6\ months}{2\ weeks}$	$\frac{2\ weeks}{2\ days}$
rely	required reliability	errors mean slight inconvenience	errors are easily recoverable	errors can risk human life
ruse	required reuse	none	multiple program	multiple product lines
sced	dictated development schedule	deadlines moved closer to 75% of the original estimate	no change	deadlines moved back to 160% of original estimate
site	multi-site development	some contact: phone, mail	some email	interactive multi-media
stor	main storage constraints (% of available RAM)	N/A	50%	95%
time	execution time constraints (% of available CPU)	N/A	50%	95%
tool	use of software tools	edit,code,debug		integrated with life cycle

Fig. 3 Parameters of the COCOMO-II effort risk model; adapted from `http://sunset.usc.edu/COCOMOII/expert_cocomo/drivers.html`. "`Stor`" and "`time`" score "N/A"" for low-end values since they have no low-end defined in COCOMO-II.

2 The Models

XOMO includes three models: COCOMO, COQUALMO, and THREAT. While the details of these models vary, they all share the variables of Figure 3. These models are summarized below (for full details, see [14]).

2.1 The COCOMO Effort model

In the COCOMO effort model, the *scale factors* of Figure 3 effect effort exponentially while *effort multipliers* effect effort linearly. COCOMO measures effort in calendar months where one month is 152 hours (and includes development and management hours). COCOMO assumes that as systems grow in size, the effort required to create them grows exponentially, i.e. $effort \propto KSLOC^x$. More precisely, COCOMO-II uses the variables of Figure 3 as follows:

$$months = a * \left(KSLOC^{\left(b+0.01*\sum_{i=1}^{5} SF_i\right)} \right) * \left(\prod_{j=1}^{17} EM_j \right) \quad (1)$$

where a and b are domain-specific parameter, and KSLOC is estimated directly or computed from a function point analysis. SF_i are the scale factors (e.g. factors such as "have we built this kind of system before?") and EM_j are the cost drivers (e.g. required level of reliability). XOMO uses the numeric ranges 1,2,3,4,5,6 to denote the COCOMO values {vl, l, n, h, vh, xh} (respectively).

2.2 The Madachy THREAT model

The Madachy THREAT model was an experiment in explicating the heuristic nature of effort estimation. It returns a

	rely= very low	rely= low	rely= nominal	rely= high	rely= very high
sced= very low	0	0	0	1	2
sced= low	0	0	0	0	1
sced= nominal	0	0	0	0	0
sced= high	0	0	0	0	0
sced= very high	0	0	0	0	0

Fig. 4 SCED-RISK: an example risk table

heuristic estimate of the threat of a schedule over run in the project. Values of 0-5 are considered to be "low threat"; 5-15 "medium threat"; 15-50 "high threat"; and 50-100 "very high threat".

Internally, the model contains dozens of tables of the form of Figure 4. Each such table adds some "threat" value to the overall project risk. These tables are read as follows. Consider the exceptional case of building high reliability systems with very tight schedule pressure (i.e. `sced=vl` or and `rely=vh` or `vh`). The COCOMO co-efficients for these ranges are 1.43 (for `sced=vl`) and 1.26 (for `rely=vh`). These co-efficients also have a threat factor of 2 (see Figure 4). Hence, a project with these two attribute ranges would contribute

$$1.43 * 1.26 * 2 = 3.6036$$

to the schedule threat.

2.3 The COQUALMO defect model

COQUALMO is in two parts:

- The *defect introduction* model looks similar to Equation 1; i.e. settings to the variables in Figure 3 map to numeric values inside COQUALMO and these are combined to report estimated number of defects per one thousand lines of delivered source code instructions.
- The *defect removal* model that represents how various tasks (peer review, execution-based testing, and automatic analysis) decrease the defect introduction estimates.

3 Data Mining

Standard numeric optimization methods may not work for XOMO. Note that our models have competing goals; e.g. lowering effort may or may not lower threats or defects. For such non-linear optimization problems, it can be useful to explore symbolic learners rather than numerical methods. XOMO uses the TAR3 data miner to find input settings that select for the better outputs.

3.1 Pre-processing the Data with "BORE"

Prior to learning, a small pre-processor was applied to the numeric outputs of COCOMO, COQUALMO, and THREAT. The BORE pre-processor converted these three outputs into the discrete class symbols ("best"; "rest") needed by TAR3.

BORE runs as follows. For each run i of the simulator, these three outputs where normalized to the range $0..1$ as follows:

$$X_i = \frac{cocomo_i - min(cocomo)}{max(cocomo) - min(cocomo)}$$

$$Y_i = \frac{coqualmo_i - min(coqualmo)}{max(coqualmo) - min(coqualmo)}$$

$$Z_i = \frac{threat_i - min(threat)}{max(treat) - min(threat)}$$

The Euclidean distance of $\{X_i, Y_i, Z_i\}$ to the ideal position of zero effort ($X_i = 0$), zero defects ($Y_i = 0$) and zero threats ($Z_i = 0$) was then computed and normalized to the range $0..1$ as follows:

$$W_i = 1 - \frac{\sqrt{X_i^2 + Y_i^2 + Z_i^2}}{\sqrt{3}}$$

W_i has the following properties:

- $0 \leq W_i \leq 1$.
- The *higher* W_i, the *better* the run.
- W_i is reduced by increasing *any* of the COCOMO effort, COQUALMO defect, or THREAT index scores. That is, improving W_i can only be achieved by decreasing *all* the effort, defects and treat scores from all the models.

To determine the "best" and "rest" values, all the W_i scores were sorted. The top 33% were then classified as "best" and the remainder as "rest". The TAR3 treatment learner was then applied to find what combination of attribute ranges selected for "best" and rejects the "rest".

3.2 Treatment Learning with "TAR3"

Treatment learning inputs a set of training examples E. Each example maps a set of attribute ranges to some class symbol; i.e. $\{R_i, R_j, ... \rightarrow C\}$ The class symbols $C_1, C_2..$ are stamped with some utility score that ranks the classes; i.e. $\{U_1 < U_2 < .. < U_C\}$. Within E, these classes occur at frequencies $F_1\%, F_2\%, ..., F_C\%$. After applying the pre-processor of the last section:

$$\begin{array}{ll} C_1 = rest & C_2 = best \\ U_1 = \ \ 2 & U_2 = \ \ 4 \\ F_1 = 66\% & F_2 = 33\% \end{array}$$

A treatment T of size X is a conjunction of attribute ranges $\{R_1 \wedge R_2 ... \wedge R_X\}$. Some subset of $e \subseteq E$ are consistent with the treatment. In that subset, the classes occur at frequencies $f_1\%, f_2\%, ...f_C\%$. A treatment learner seeks the seek smallest T which most changes the weighted sum of the utilities times frequencies of the classes. Formally, this is called the $lift$ of a treatment:

$$lift = \frac{\sum_C U_C f_C}{\sum_C U_C F_C}$$

For example, consider the log of golf playing behavior seen in Figure 6. In that log, we only play *lots* of golf in

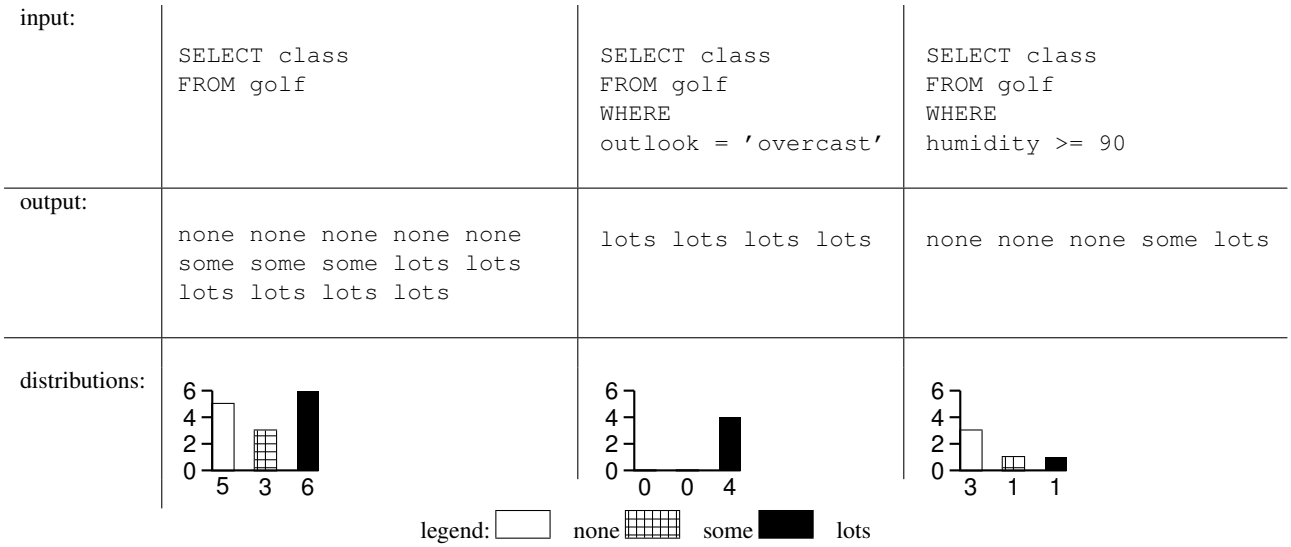

Fig. 5 TAR3: Class distributions selected by different conditions in Figure 6.

outlook	*temp(^{o}F)*	*humidity*	*windy?*	*class*
sunny	*85*	*86*	*false*	*none*
sunny	*80*	*90*	*true*	*none*
sunny	*72*	*95*	*false*	*none*
rain	*65*	*70*	*true*	*none*
rain	*71*	*96*	*true*	*none*
rain	*70*	*96*	*false*	*some*
rain	*68*	*80*	*false*	*some*
rain	*75*	*80*	*false*	*some*
sunny	*69*	*70*	*false*	*lots*
sunny	*75*	*70*	*true*	*lots*
overcast	*83*	*88*	*false*	*lots*
overcast	*64*	*65*	*true*	*lots*
overcast	*72*	*90*	*true*	*lots*
overcast	*81*	*75*	*false*	*lots*

Fig. 6 TAR3: Playing golf.

$\frac{6}{5+3+6} = 43\%$ of cases. To improve our game, we might search for conditions that increases our golfing frequency. Two such conditions are shown in the `WHERE` test of the select statements in Figure 5. In the case of `outlook= overcast`, we play *lots* of golf all the time. In the case of `humidity` $\leq$ `90`, we only play *lots* of golf in 20% of cases. So one way to play lots of golf would be to select a vacation location where it was always overcast. While on holidays, one thing to watch for is the humidity: if it rises over 90%, then our frequent golf games are threatened.

The tests in the `WHERE` clause of the select statements in Figure 5 is a treatment. Classes in treatment learning get a score U_C and the learner uses this to assess the class frequencies resulting from *applying a treatment* (i.e. using them in a `WHERE` clause). In normal operation, a treatment learner does *controller learning* that finds a treatment which selects for better classes and reject worse classes By reversing the scoring function, treatment learning can also select for the worse classes and reject the better classes. This mode is called *monitor learning* since it finds the thing we should most watch for. In the golf example, *outlook = 'overcast'* was the controller and $humidity \geq 90$ was the monitor.

Formally, treatment learning is a weighted-class minimal contrast-set association rule learner. The treatments are associations that occur with preferred classes. These treatments serve to contrast undesirable situations with desirable situation where more of the outcomes are favorable. Treatment learning is different to other contrast set learners like STUCCO [1] since those other learners don't focus on minimal theories.

Conceptually, a treatment learner explores all possible subsets of the attribute ranges looking for good treatments. Such a search is impractical in practice so the art of treatment learning is quickly pruning unpromising attribute ranges. This study uses the TAR3 treatment learner [9] that uses stochastic search to find its treatments.

4 Iterative Treatment Learning

Figure 1 describes an iterative process of applying a data miner over and over again to progressively refine input ranges into a model. These refinements can be viewed as a tree of attribute ranges:

- The root of the tree describes some initial ranges;
- Sub-trees store narrower and narrower ranges.

For example, the analysis in this paper resulted in the tree of Figure 7. In such a tree, nodes inherit the ranges of their parents *unless* they elect to restrain those ranges. For example:

- "system" stores default ranges for COCOMO, COQUALMO, and THREAT.

```
system      #⇐ see Figure 8
| ares      #⇐ see Figure 9
| | 1
| | | 2
| | | | 3
| | | | | 4
| | | | | | 5
| | | | | | | 6
| | | | | | | | 6a
| | | | | | | | 6b
| | | | | | | | 6c
| | | | | | | | 6d
| | | | | | | | 6e
| | | | | | | | 6f
```

Fig. 7 A range tree.

```
 1 # constants
 2 A     is real 2.25 3.25
 3 B     is real 0.9  1.1
 4 ksloc is int 2 10000
 5
 6 automated_analysis          is int 1 6
 7 peer_reviews                is int 1 6
 8 execution_testing_and_tools is int 1 6
 9
10 # scale factors
11 prec  is int 1 5
12 flex  is int 1 5
13 resl  is int 1 5
14 team  is int 1 5
15 pmat  is int 1 5
16
17 # effort multipliers
18 time  is int 3 6
19 stor  is int 3 6
20 data  is int 2 5
21 pvol  is int 2 5
22 ruse  is int 1 5
23 rely  is int 1 5
24 docu  is int 1 5
25 acap  is int 1 5
26 pcap  is int 1 5
27 pcon  is int 1 5
28 aexp  is int 1 5
29 plex  is int 1 5
30 ltex  is int 1 5
31 tool  is int 1 5
32 sced  is int 1 5
33 cplx  is int 1 6
34 site  is int 1 6
```

Fig. 8 The ranges in "system".

- "ares" restrains those ranges to just those relevant to the ARES GNC model for ARES1.
- Nodes "1,2,3,4,5,6" store restraints learned via TAR3 applying Figure 1, six times.
- Nodes "6a,6b,6c,6d,6e,6f" store some what-queries that explore some manually selected variants to node "6". These variants explore the merits of using increasing levels of automated analysis without performing execution testing.

Figure 8 and Figure 9 shows the contents of the "system" and "ares" nodes in the tree of ranges. Note that the default "system" ranges are not point estimates. Unless some subtree restrains a range, the Monte Carlo simulator will select randomly from some pre-defined range. For example, lines 2&3 of Figure 8 show ranges for the two COCOMO effort model *calibration parameters*. Standard practice is to tune COCOMO by tuning these values using local data. When such local data is missing (e.g. in this study), we use Monte

```
 1 system with ares
 2 ksloc just  75 125
 3 prec just 3 5
 4 flex = 3
 5 resl = 4
 6 team = 3      # typo. was 2
 7 pmat just 4 5 # typo. was 5 6
 8 rely = 5
 9 data = 4
10 cplx = 4
11 ruse = 4
12 docu just 3 4
13 time = 3
14 stor = 3       # typo. was 2
15 pvol = 3
16 acap = 4
17 pcap = 3
18 pcon = 3
19 aexp = 4
20 plex = 4
21 ltex just 2 5
22 tool = 5
23 site = 6
24 sced just 2 4
```

Fig. 9 The ranges in "ares". The lines marked as "typo" are those reflecting minor range typos in the ARES document describing its COCOMO ranges.

Carlo simulation to explore a wide range of possible calibration parameters.

Note also that "ares" restrains some, but not all, of the ranges in "system". For example, "ares"'s KSLOC guesstimates (75 to 125) are much less than for arbitrary "system"'s (2 to 10,000). However, there are 29 ranges in Figure 8 and only 23 in Figure 9; that is, all we know about ARES is not enough to generate deterministic predictions about the effort, defects, and threats to ARES software.

5 Running the System

To generate our results, XOMO was run as follows:

```
xomo         \
  -S $RANDOM  \
  -l           \
  -R 1000       \
  -d system.dat  \
  -p ares.dat     \
  -P are           \
  -s '<$defects,<$effort,<$threats,?$A,?$B,$ksloc' > xomo.out
```

The command can be read as follows:

`-S $RANDOM`: use a random seed for the simulations.

`-R 1000`: perform 1000 Monte Carlo Simulations

`-l`: minor detail- print a header on the output;

`-d $system.dat`: load default ranges from Figure 8;

`-p $ares.dat`: load the ARES ranges Figure 9;

`-P ares`: focus the simulation on ARES;

`-s '<$defects....'`: a string specifying the goal statement; i.e. minimize defect, effort, and treats.

Prior to calling TAR3, the results on the above command (stored in the *tmp* file) must be divided into "best" and "rest"

```
bore N=0.33 Pass=1 xomo.out \
            Pass=2 xomo.out  \
            Pass=3 xomo.out > xomo.data
```

=execution testing and tools	flex	stor	aexp	docu	site	plex	cplx	ltex	peer reviews	$ksloc	rely	data	ruse	acap	?$A	sced	automated analysis	pcon	?$B	pcap	tool	time	prec	pvol	resl	team	pmat	<$risk	<$effort	<$defects
1	3	3	4	4	6	4	4	5	6	116	5	4	4	4	3.10408	3	6	3	0.915776	3	5	3	4	3	4	3	5	0.68	345.87	1.67
1	3	3	4	4	6	4	4	5	6	78	5	4	4	4	2.80696	3	6	3	0.92094	3	5	3	3	3	4	3	4	0.68	238.43	2.00
1	3	3	4	4	6	4	4	5	6	118	5	4	4	4	2.51974	3	6	3	1.09107	3	5	3	5	3	4	3	5	0.68	621.76	1.55
1	3	3	4	4	6	4	4	5	6	118	5	4	4	4	2.808	3	6	3	1.09649	3	5	3	3	3	4	3	4	0.68	862.15	2.00
1	3	3	4	3	6	4	4	5	6	118	5	4	4	4	3.23396	3	6	3	1.04294	3	5	3	5	3	4	3	4	0.68	615.55	1.88
1	3	3	4	3	6	4	4	5	6	90	5	4	4	4	2.59488	3	6	3	1.01176	3	5	3	3	3	4	3	4	0.68	348.81	2.16
1	3	3	4	3	6	4	4	5	6	94	5	4	4	4	2.54054	3	6	3	0.9104	3	5	3	3	3	4	3	4	0.68	226.74	2.16
1	3	3	4	4	6	4	4	5	6	97	5	4	4	4	2.38838	3	6	3	1.02511	3	5	3	5	3	4	3	5	0.68	343.77	1.55
1	3	3	4	4	6	4	4	5	6	125	5	4	4	4	2.33558	3	6	3	1.02301	3	5	3	3	3	4	3	4	0.68	540.69	2.00
...																														

Fig. 10 XOMO output.

=execution testing and tools	flex	stor	aexp	docu	site	plex	cplx	ltex	peer reviews	$ksloc	rely	data	ruse	acap	?$A	sced	automated analysis	pcon	?$B	pcap	tool	time	prec	pvol	resl	team	pmat	best or rest?
1	3	3	4	4	6	4	4	5	6	77	5	4	4	4	3.16394	3	6	3	0.900243	3	5	3	5	3	4	3	5	_1
1	3	3	4	3	6	4	4	5	6	75	5	4	4	4	2.98155	3	6	3	1.09603	3	5	3	3	3	4	3	4	_0
1	3	3	4	3	6	4	4	5	6	84	5	4	4	4	2.4412	3	6	3	1.01243	3	5	3	3	3	4	3	4	_0
1	3	3	4	3	6	4	4	5	6	102	5	4	4	4	3.1926	3	6	3	0.971668	3	5	3	3	3	4	3	4	_0
1	3	3	4	3	6	4	4	5	6	117	5	4	4	4	2.33322	3	6	3	0.934947	3	5	3	4	3	4	3	4	_0
1	3	3	4	4	6	4	4	5	6	115	5	4	4	4	2.95164	3	6	3	0.990819	3	5	3	5	3	4	3	5	_1
1	3	3	4	3	6	4	4	5	6	122	5	4	4	4	2.51071	3	6	3	0.957017	3	5	3	5	3	4	3	5	_1
1	3	3	4	4	6	4	4	5	6	117	5	4	4	4	3.01523	3	6	3	0.961085	3	5	3	4	3	4	3	4	_0
1	3	3	4	3	6	4	4	5	6	112	5	4	4	4	2.48963	3	6	3	1.07945	3	5	3	4	3	4	3	4	_0
1	3	3	4	3	6	4	4	5	6	117	5	4	4	4	2.77276	3	6	3	1.00916	3	5	3	4	3	4	3	5	_0
...																												

Fig. 11 BORE output. In the last column, _0 denotes "rest" and _1 denotes "best".

For example, Figure 10 shows a typical $xomo.out$ file and Figure 11 shows the result of BORE replacing the last three columns with "best" or "rest".

Once the data has been BOREd, it is passed to TAR3 as follows:

```
tar3 xomo
```

With the above call, TAR3 expects to find three files:

1. $xomo.data$: generated by BORE and shown in Figure 11;
2. $xomo.names$: a data dictionary for the data file, shown in Figure 12;
3. $xomo.cfg$: some control settings, shown in Figure 13.

$Xomo.names$ is almost self-explanatory. One non-obvious feature is on line one where the order of the classes tells TAR3 what to seek and what to avoid. In a TAR3 $names$ file, the classes are weighted left to right 2,4,8,16,etc. Hence, in Figure 12, _1 (i.e. "best") is the preferred class.

$Xomo.cfg$ requires some explanation:

- $Granularity$ controls how the continuous ranges are divided into bins. For reporting purposes, an odd number for $Granularity$ is best since (e.g.) a $Granularity$ of 5 can be reported as "3 means no change, 4 and 2 means some changes up and down, and 5 and 1 mean larger changes up and down". $Granularities$ over 7 are rarely useful and for problematic data sets (like the XOMO data), this number can go as low as 2.
- TAR3 only reports the top $MaxNumber$ of treatments (in this case, 10).

```
_0,_1

execution_testing_and_tools :  1,2,3,4,5,6.
flex  :  1,2,3,4,5,6.
stor  :  1,2,3,4,5,6.
aexp  :  1,2,3,4,5,6.
docu  :  1,2,3,4,5,6.
site  :  1,2,3,4,5,6.
plex  :  1,2,3,4,5,6.
cplx  :  1,2,3,4,5,6.
ltex  :  1,2,3,4,5,6.
peer_reviews  :  1,2,3,4,5,6.
ksloc  :  continuous.
rely  :  1,2,3,4,5,6.
data  :  1,2,3,4,5,6.
ruse  :  1,2,3,4,5,6.
acap  :  1,2,3,4,5,6.
A     :  continuous.
sced  :  1,2,3,4,5,6.
automated_analysis  :  1,2,3,4,5,6.
pcon  :  1,2,3,4,5,6.
B  :  continuous .
pcap  :  1,2,3,4,5,6.
tool  :  1,2,3,4,5,6.
time  :  1,2,3,4,5,6.
prec  :  1,2,3,4,5,6.
pvol  :  1,2,3,4,5,6.
resl  :  1,2,3,4,5,6.
team  :  1,2,3,4,5,6.
pmat  :  1,2,3,4,5,6.
```

Fig. 12 xomo.names.

```
granularity:   2
maxNumber:    10
maxSize :     10
randomTrials: 100
futileTrials:  5
bestClass:    20\%
```

Fig. 13 xomo.cfg

- When composing a treatment, TAR3 will build constraints of up to $MaxSize$ items. The upper bound on this number is the number of attributes but, in practice, a MaxSize of 5-10 often suffices.
- $RandomTrils$ sets the number of trial we perform before we $pause$ to look for new best treatments.
- $FutileTrails$ is the number of allowed pauses (so the total number of trials is 100*5).
- TAR3 rejects any treatment that contains less than $BestClass$ percentage of the best class.

The settings in Figure 13 could be improved but, as we shall see, they suffice for the XOMO data.

6 Results

Figure 14 shows the effects of applying Figure 1 to ARES1. After each round, TAR3 learns treatments and the best treatment is selected to restrain the Monte Carlo simulation of the next round. At each round, the "best/rest" discretization described above was repeated so "best" was the top 33% seen in each round:

The first round decision was $sced = 3$; i.e. set the schedule pressure to the mid-point of its ranges $\{2,3,4\}$ (as shown on the last line of Figure 9. The second round decision was to apply maximum effort to execution-based testing. As shown right-hand-side plots of Figure 1, these failed to reduce mean defects or effort. Round 1 and 2 did achieve a small reduction in the threat index. However, recall that the THREAT model classifies threat indexes less than 6 as "low threat". Hence, the round 1&2 reduction of THREAT from 2 to 0.5 is not particularly interesting.

round1:
sced = 3

round2:
execution_testing_and_tools = 6

round3:
peer_reviews = 6

round4:
automated_analysis = 6

round5:
ltex = 5

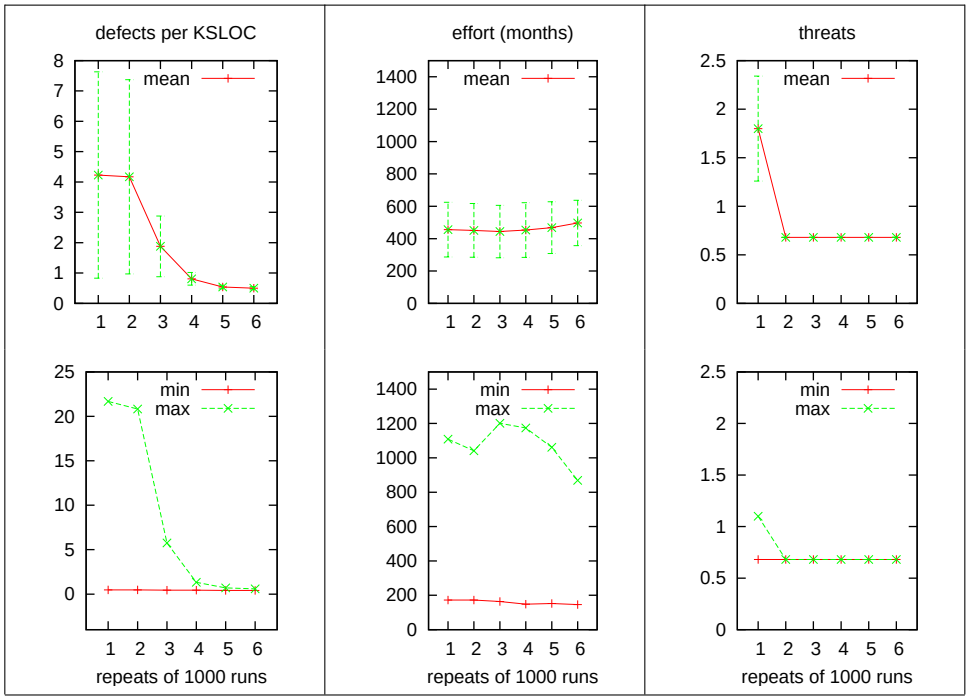

Fig. 14 Results. In the top row, the green error bars denote ± one standard deviation about the mean.

round	sced	peer reviews	ltex	execution testing and tools	automated analysis	defects			
						min	mean	max	sd
6a	3	6	5	1	1	3.0	3.6	4.25	3.6
6b	3	6	5	1	2	2.9	3.5	4.09	3.5
6c	3	6	5	1	3	2.6	3.1	3.6	3.1
6d	3	6	5	1	4	2.2	2.6	3.0	2.6
6e	3	6	5	1	5	1.7	2.1	2.42	2
6f	3	6	5	1	6	1.6	1.8	2.2	1.8
6	3	6	5	6	6	0.42	0.5	0.59	0.5

Fig. 15 Effects of increasing automated analysis.

At round 3, TAR3 proposed maximizing the effort on peer reviews and this had dramatic impact on the defects: the mean value halved and the maximum value dropped 75%.

At round 4, TAR4 proposed maximizing the effort on automated analysis. This round halved again the mean estimated number of defects (from two to one) and reduced the maximum value by a further 80%.

At round 5, the best treatment TAR3 found was to increase the language and tools experience; i.e. $ltex = 5$. Since the improvements seen from round 4 to round 5 are negligible, cyclic learning was terminated.

Our prior XOMO-based analysis of a space shuttle subsystem offered far more dramatic results than Figure 14 [14]. That prior study found ways to:

- Reduce the residual defects per KSLOC by 85%;
- Halve the threat of schedule over-run;
- Decrease the development effort to nearly half of its original value.

Why did that study produce more impressive results than this one? Our view is that the issue is in the problem domain and not in the analysis method. In this study, 24 attributes were set and only a handful could be varied. In the former study, our data miner was given far more opportunity to improve the project.

7 Discussion

The results of Figure 14 shows interesting subtleties:

- Despite certain uncertainties in the domain (e.g. the exact value of the calibration parameters A, B), note that the effort and threat mins/max/mean variables are quite stable. That is, the ARES system displays much stability in its process properties.
- The merits of execution-based testing, by itself, seem very small for this software. Note that the addition of execution-based testing (in round 2) barely changed the defect rate.

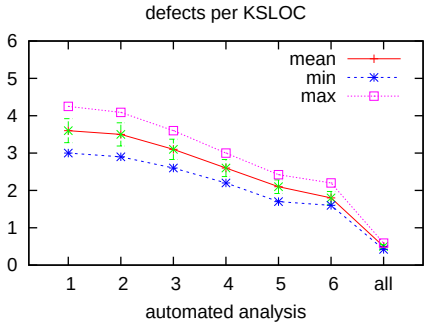

Fig. 16 Figure 15, plotted. Green error bars denote $\pm$ one standard deviation about the mean.

- On the other hand, peer reviews seem most powerful for this software. Note that the greatest defect rate reduction occurred when peer reviews were added in round 3.

The ability to make fine-grained trade-offs between alternate technologies is one of the main advantages of the XOMO technology. For example, Figure 15 and Figure 16 shows an XOMO study with ARES where:

- peer review usage was maximized;
- execution-based testing was minimized;
- and automatic analysis was increased from very low to very high levels.

This represents a very common situation at NASA- given limited access to hardware test beds, IV&V analysts perform very limited execution-based testing. In such a situation, peer reviews are widely used (since they are relatively cheap to perform). In this context, it might be necessary to make a business case to buy some automatic analysis tool. Figure 15 and Figure 16 show the effects of only changing the level of automatic analysis while holding other defect removal methods steady. Automatic analysis can be cheap to perform (e.g. static code parsers) and sometimes can be conducted very early in the life cycle (e.g. lightweight formal models of requirements).

Figure 16 focuses on how the level of automatic analysis effects defect estimates. Note that automatic analysis by itself does not get rid of all the defects. But, in the absence of execution-based testing tools, it can half the mean predicted number of defects.

8 Conclusion

This paper began with three questions:

1. What level of risk is acceptable for this project?
2. How much risk does a project have?
3. How to reduce #2 to #1?

XOMO lets us explore #2 with XOMO & BORE & TAR3 to find methods for #3. As to question #1, this answer is domain-specific and depends on manner factors such as the goal & cost of the mission; the degree of innovation; and,

most importantly, the presence of humans on-board the mission. But once the appropriate level of risk is determined, our tools can find ways to explore business knowledge to adjust a project towards an acceptable level of risk.

References

1. S. Bay and M. Pazzani. Detecting change in categorical data: Mining contrast sets. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999. Available from http://www.ics.uci.edu/~pazzani/Publications/stucco.pdf.
2. B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani, and C. Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.
3. T. Burkleaux, T. Menzies, and D. Owen. Lean = (lurch+tar3) = reusable modeling tools. In *Proceedings of WITSE 2005*, 2004. Available from http://menzies.us/pdf/04lean.pdf.
4. E. Chiang and T. Menzies. Simulations for very early lifecycle quality evaluations. *Software Process: Improvement and Practice*, 7(3-4):141–159, 2003. Available from http://menzies.us/pdf/03spip.pdf.
5. S. L. Cornford, J. D. M. S. Feather, J. Salcedo, and T. Menzies. Optimizing spacecraft design optimization engine development: Progress and plans. In *Proceedings of the IEEE Aerospace Conference, Big Sky, Montana*, 2003. Available from http://menzies.us/pdf/03aero.pdf.
6. M. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002. Available from http://menzies.us/pdf/02re02.pdf.
7. M. Fisher and T. Menzies. Learning iv&v strategies. In *HICSS'06*, 2006. Available from http://menzies.us/pdf/06hicss.pdf.
8. D. Geletko and T. Menzies. Model-based software testing via treatment learning. In *IEEE NASE SEW 2003*, 2003. Available from http://menzies.us/pdf/03radar.pdf.
9. Y. Hu. Treatment learning, 2002. Masters thesis, Unviersity of British Columbia, Department of Electrical and Computer Engineering. In preperation.
10. T. Menzies and Y. Hu. Constraining discussions in requirements engineering. In *First International Workshop on Model-based Requirements Engineering*, 2001. Available from http://menzies.us/pdf/01lesstalk.pdf.
11. T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from http://menzies.us/pdf/03tar2.pdf.
12. T. Menzies and J. Kiper. Better reasoning about software engineering activities. In *ASE-2001*, 2001. Available from http://menzies.us/pdf/01ase.pdf.
13. T. Menzies, D. Raffo, S. on Setamanit, Y. Hu, and S. Tootoonian. Model-based tests of truisms. In *Proceedings of IEEE ASE 2002*, 2002. Available from http://menzies.us/pdf/02truisms.pdf.
14. T. Menzies and J. Richardson. Xomo: Understanding development options for autonomy. In *COCOMO forum, 2005*, 2005. Available from http://menzies.us/pdf/05xomo_cocomo_forum.pdf.
15. T. Menzies and J. Richardson. Making sense of requirements, sooner. *IEEE Computer*, October 2006. Available from http://menzies.us/pdf/06qrre.pdf.
16. T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from http://menzies.us/pdf/00ase.pdf.
17. D. Owen, T. Menzies, and B. Cukic. What makes finite-state models more (or less) testable? In *IEEE Conference on Automated Software Engineering (ASE '02)*, 2002. Available from http://menzies.us/pdf/02moretest.pdf.
18. B. C. Tim Menzies, David Owen. You seem friendly, but can i trust you? In *Formal Aspects of Agent-Based Systems*, 2002. Available from http://menzies.us/pdf/02trust.pdf.