RST milestone 1.1.5.9: Applying trade space analysis to recommend CEV/CLV options for SW capabilities and development of processes and tools

Tim Menzies

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

Abstract We seek an AI agent that can negotiate the trade space within the design of modern complex systems such as NASA's new CEV/CLV rockets. The conclusions of the agent must be comprehensible to busy humans engaged in elaborate discussions regarding project options. Hence, those conclusions should be clear to read and clearly useful. As an on-going task in this project, we are assessing the merits of the TAR3 *treatment learning* for such an agent.

This paper presents several case studies where TAR3 successfully found succinct recommendations that clearly effected the development effort, number of defects, and threats to the project plan.

The technology described here integrates into the RST test bed via an interface called *slot trees* (described in this document).

Acknowledgments 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).

Contents

List of Figures

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.

Technically, risk reduction is a *trade-space* exploration exercise. Given a set of options generated by a set of models, managers must "walk" (metaphorically) around that space looking for decisions that most improve the current situation. Our trade-spaces are defined by three models:

– The COCOMO effort estimation model [3, p29-57];
– The COQUALMO defect model [3, p254-268];
– The THREAT schedule threat model [3, 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 large trade-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. $ranges_{i+1} \subseteq ranges_i$.

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 treatment learner [17] is called XOMO (pronounced "x-o-mow") [20]. XOMO uses TAR3 since this learner 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" [20].

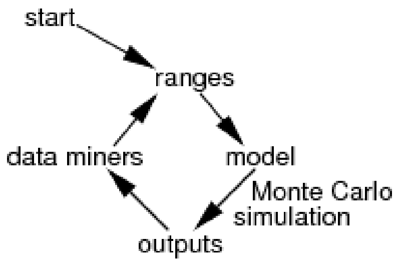


Fig. 1 Cyclic learning

This paper applies XOMO to a variety of models taken from the Reliable Software Test bench under development at the AMES Research Center. For example, one item in RST is ARES, a model of the guidance and navigation control (GNC) system of ARES1 (formerly known as the CEV). Our methods supported a fine-grained analysis of the merits of various process options: e.g. 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.

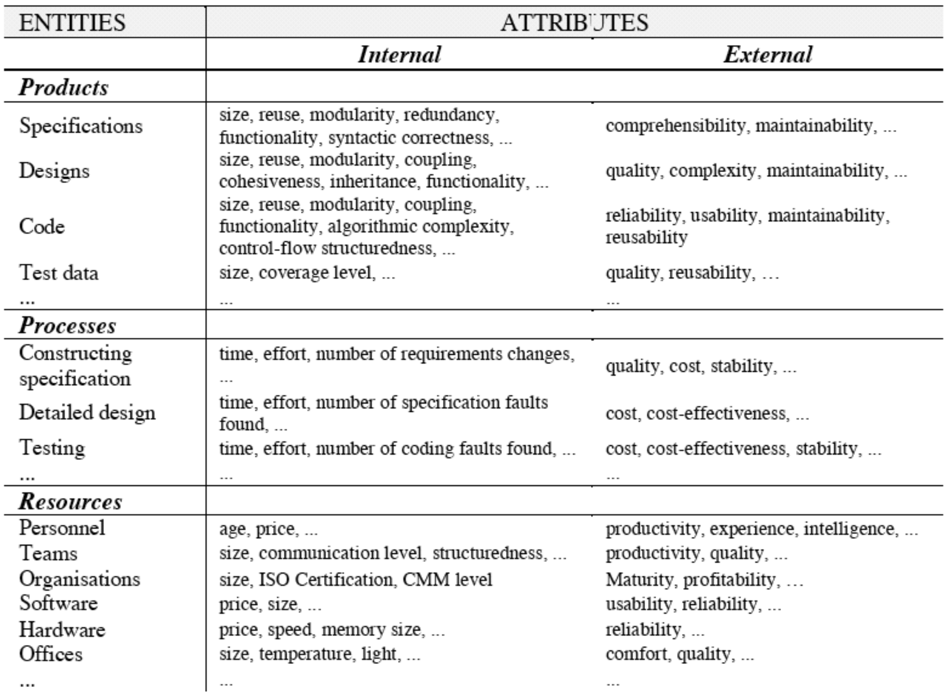
Within NASA, Figure 1 has been applied to:

– Spacecraft design, where fitter means "covers more requirements and reduces most risk, costs the least" [6, 7].
– Software process control using:
  – A Chung-Mypolopous soft-goal graph where fitter means "cover more non-functional requirements" [5].
  – The COCOMO and THREAT models where fitter means "lower effort and fewer threats" [23];
  – The COCOMO and THREAT and COQUALMO models where fitter means "lower effort and fewer threats and lower defects" [20];
  – Qualitative inference diagrams where fitter means "higher quality" [21].
  – The IV&V "Silap" model that selects work break down structures for V&V where fitter means "lower risks" [10];

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

– Circuit design, with fitter being "more bulbs shine" [16].
– Finite state machines, where fitter means "better chances of covering transitions" [24, 27].
– Economic policy where fitter means "longer human life" [11];
– Whiskey production, where fitter means "more alcohol" [4];
– Software process control using:
  – a CMM level 2 model, where fitter mean "chance of reaching a lost cost project" [18];
  – Discrete event simulation where fitter was defined by a utility function combining quality, expense, and development time [19].

Fig. 2 Prior applications of Figure 1.



ENTITIES	ATTRIBUTES	
	Internal	***External***
Products		
Specifications	size, reuse, modularity, redundancy, functionality, syntactic correctness, ...	comprehensibility, maintainability, ...
Designs	size, reuse, modularity, coupling, cohesiveness, inheritance, functionality, ...	quality, complexity, maintainability, ...
Code	size, reuse, modularity, coupling, functionality, algorithmic complexity, control-flow structuredness, ...	reliability, usability, maintainability, reusability
Test data	size, coverage level, ...	quality, reusability, ...
...	...	...
Processes		
Constructing specification	time, effort, number of requirements changes, ...	quality, cost, stability, ...
Detailed design	time, effort, number of specification faults found, ...	cost, cost-effectiveness, ...
Testing	time, effort, number of coding faults found, ...	cost, cost-effectiveness, stability, ...
...	...	...
Resources		
Personnel	age, price, ...	productivity, experience, intelligence, ...
Teams	size, communication level, structuredness, ...	productivity, quality, ...
Organisations	size, ISO Certification, CMM level	Maturity, profitability, ...
Software	price, size, ...	usability, reliability, ...
Hardware	price, speed, memory size, ...	reliability, ...
Offices	size, temperature, light, ...	comfort, quality, ...
...	...	...

Fig. 3 Types of software measures. From [8].

2 Background

2.1 Related Work

Fenton and Pfleeger [9] advise that measurements from software projects divide into the three groups of Figure 3:

– *Process measures*: describing *how* the software is built;
– *Product measures*: describing *what* is being built;
– *Resource measures*: describing *who* is doing the building and *which* tools are being used in the process.

Each of these measures have their own kind of risks:

– Process risks can arise when (say) the methods are applied to a a project;
– Product risks can arise when (say) a faulty piece of software is delivered;
– Resource risks can arise when (say) developers are given too little time to build a complex device.

Risk reduction methods for all possible process, product, and resource risks are beyond the scope of this paper. Previously [23] we have followed the lead of Madachy [14], and have focused on the risks that can be expressed using the COCOMO ontology Figure 4. One advantage of COCOMO (and this is why we use it) is that unlike many other costing models such as PRICE-S [25], SEER-SEM [13] or SLIM [26], COCOMO is an *open model* so complete information is available on its internal structure [2, 3]. Also, on NASA projects, COCOMO similar results to other cost models. For example, in comparative studies of SEER and COCOMO-II for CEV, similar cost estimates were generated.

scale factors	prec:	have we done this before?
	flex:	development flexibility
	resl:	any risk resolution activities?
	team:	team cohesion
	pmat:	process maturity
upper	acap:	analyst capability
	pcap:	programmer capability
	pcon:	programmer continuity
	aexp:	analyst experience
	pexp:	programmer experience
	ltex:	language and tool experience
	tool:	tool use
	site:	multiple site development
	sced:	length of schedule
lower	rely:	required reliability
	data:	secondary memory storage requirements
	cplx:	program complexity
	ruse:	software reuse
	docu:	documentation requirements
	time:	runtime pressure
	stor:	main memory requirements
	pvol:	platform volatility

Fig. 4 The COCOMO ontology. Development effort is *exponentially*, *linearly*, and *negatively linearly* proportional to the *scale factors*, *upper* terms, and *lower* terms (respectively).

To some degree, the COCOMO ontology addresses issues of process,product, and resource metrics. For example, the COCOMO terms of Figure 4 can be divided as follows:

– *Resl, pmat, site* and *, sced* are process measures;
– *Prec, tool, rely, data, cplx, ruse, docu, time, stor* and *pvol* are product measures;
– *Flex, team, acap, pcap, pcon, aexp* and *ltex* are resource measures.

Nevertheless, the standard COCOMO does not address many other risk issues. The COCOMO team recognizes this and new terms were added for the COQUALMO defect prediction model[2]:

- COQUALMO recognizes project phases: *requirements*, *design* and *coding*;
- COQUALMO also divides defect removal tools into *automated analysis, peer reviews* and *execution testing and tools*.

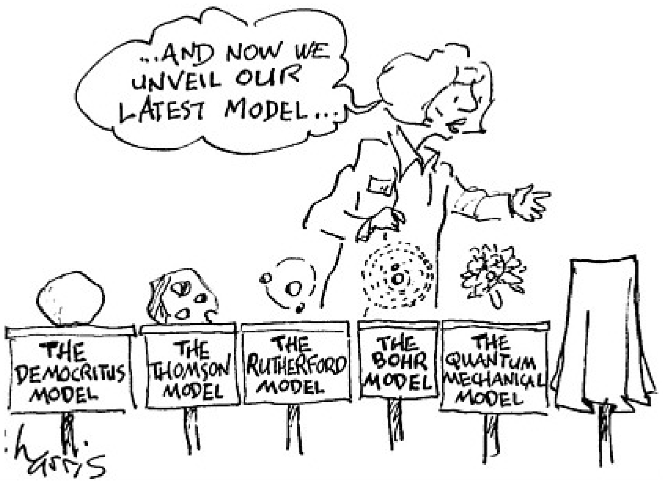
2.2 *Are our Models "Correct"?*

One drawback with our results is that they come from simulation and not from empirical observations of real world development teams applying the policy decisions made by the learners. If our models are wrong (e.g. poorly calibrated) then our results are suspect.

Our methodology partially addresses this concern. For example, the COCOMO effort model contains two calibration parameters and the above results hold for simulations across the space of possible calibrations. That is, Monte Carlo plus data mining can find stable conclusions within the space of possibilities (this is a conclusion we have made elsewhere, many times [5, 15–17, 19, 22, 23]).

Nevertheless, simulations across the space of options will never give the right answers if that model is fundamentally flawed; e.g. important domain factors are missing from the model. This is a problem with all model-based reasoning: if the model is wrong then the reasoning is wrong as well.

However, as George Box says, "all models are wrong but some are useful". Certainly this has been the recent experience in physics. Over the last 100 models, numerous revisions to the atomic theory of matter have been proposed:



Each new model was wrong since it was superseded by a newer model. But each new model was useful in the sense that it explained more effects than the previous model.

The lesson here is that committing to a model of the current best understanding of a phenomenon is good practice, even if that model is not "correct" in some absolute sense. And once that new model is generated, it is right and proper that it be exercised, criticized, and improved. In the next few months, the XOMO models will be used in panel sessions were experts will convene to debate cost and risk models for autonomous NASA software. At those sessions, it is expected that the XOMO models will be critiqued and extensively revised.

During those panels, it is important that the experts' time is put to best use. NASA's top software and hardware experts are scarce and it will take a significant administrative effort to collect them all together at the same place and at the same time. It is therefore vital that no time be wasted in discussing irrelevancies. The XOMO toolkit can be used to quickly prune debates about relatively unimportant issues. The panel moderator could (gently) guide the discussion onto other matters if the matters under debate have little effect on the model behaviors. Similar, the moderator could ask XOMO for the next most important issue to discuss (that issue would be the one that most changes the current model's behavior).

3 Model Inputs: Slot Trees

XOMO inputs a set of ranges, and some constraints on those ranges. It then executes COCOMO, COQUALMO, and THREAT by selecting inputs drawn from those constrained ranges. The resulting data set is then passed to our learner. After learning, XOMO outputs "better" constraints on those ranges; i.e. the subset of the constraints that most select for (say) fewer defects, lower cost, fewer bugs.

XOMO's inputs are expressed the *slot tree* format. A *slot tree* is a tree of frames. Each frame contains slots. The root of the tree stores maximal ranges for each variable and sub-frames stores restrictions on those values. In the current slot tree language, ranges can be defined using one of four forms:

- X is int Min Max
- X is real Min Max
- X is normal Mean Sd
- X is of item1 item2 item3 ....

For example:

```
age    is int    0 120
angle  is real   0 6.28
weight is normal 20 4
city   is of     oxford london paris
```

This root frame defines:

- *age* to be an integer ranging from zero to 120;
- *angle* to be a real number ranging from 0 to 6.28;
- *weight* to be a Gaussian with a mean of 20 and standard deviation of 4;
- and *city* to be one of *oxford*, *london*, or *paris*.

The root frame for XOMO is shown in Figure 5 (note that it offers default ranges from the COCOMO terms of Figure 4).

In a slot tree, frames can have sub-frames and, recursively, sub-frames can have sub-sub-frames. This tree of sub-frames can have two applications. Firstly, it can be used to

[2] http://csse.usc.edu/research/coqualmo/

```
 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. 5 The ranges in "system".

```
system with ares
ksloc just  75 125
prec just 3 5
flex = 3
resl = 4
team = 3
pmat just 4 5
rely = 5
data = 4
cplx = 4
ruse = 4
docu just 3 4
time = 3
stor = 3
pvol = 3
acap = 4
pcap = 3
pcon = 3
aexp = 4
plex = 4
ltex just 2 5
tool = 5
site = 6
sced just 2 4
```

Fig. 6 The ranges in "ares". ranges.

```
function Effort() {
  return A() * Ksloc() ^ E()   * Rely()* Data()* Cplx()*
         Ruse()* Docu()* Time()* Stor()* Pvol()* Acap()*
         Pcap()* Pcon()* Aexp()* Plex()* Ltex()* Tool()*
         Site()* Sced()
}

function E() {
  return B() + 0.01*(Prec() + Flex()
             + Resl() + Team() + Pmat())
}
```

Fig. 8 COCOMO: computing effort.

succinctly define a software project. For example, Figure 6 describes a NASA project as a specialization of the general COCOMO ranges of Figure 5. Note that Figure 6 does not need to mention all the COCOMO variables- variables that skipped in a leaf are inherited from higher in the slot tree.

Leaf frames offer restrictions to root ranges (but can't define new ranges). Leaf frames are connected to parent frames with the keyword $with$. Restrictions can be defined using one of two forms:

- X just A B
- X = Value

For example, Figure 6 shows some restrictions to Figure 5.

Unless some sub-tree restrains a range, the Monte Carlo simulator will select randomly from some pre-defined range. For example, lines 2&3 of Figure 5 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 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 5 and only 23 in Figure 6; that is, all we know about ARES is not enough to generate deterministic predictions about the effort, defects, and threats to ARES software.

The other role of slot trees is iterative learning. XOMO's data miners run in cycles. In each cycle, a few hundred Monte Carlo simulations are followed by some learning which, in turn, outputs some new restraints. The outputs of each cycle can be a new slot tree leaf.

4 Models

At runtime, XOMO pulls values from slot trees and passes them to three models.

4.1 The COCOMO Effort Model

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 variable of Figure 7 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). Scale factors have

	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. 7 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.

an exponential impact on software cost while effort multipliers have a linear impact.

Figure 8 shows XOMO implementation of the COCOMO effort equation. Values such as (e.g.) `flex=1` get converted to numerics as follows. First, the integers $\{1, 2, 3, 4, 5, 6\}$ are converted to the symbols {vl, l, n, h, vh, xh} (respectively) representing very low, low, nominal, high, very high, and extremely high. Next, these are mapped into the look-up table of Figure 9.

4.2 THREATS: a Heuristic Risk Model

The Madachy Heuristic schedule THREAT model was an experiment in explicating the heuristic nature of effort estimation. It returns a heuristic estimate of the chances 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". Studies with the COCOMO-I project database have shown that the Madachy THREAT index correlates well with $\frac{months}{KDSI}$ (where KDSI is thousands of delivered source lines of code) [14].

Internally, the model contains dozens of tables of the form of Figure 10. Each such table adds some "threatiness" value to the overall project threat. 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`). Recalling Figure 9, the COCOMO coefficients 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 10). Hence, a project with these two attribute ranges would contribute 1.43*1.26*2=3.6036 to the schedule threat.

	vl	l	n	h	vh	xh
Scale factors:						
flex	5.07	4.05	3.04	2.03	1.01	
pmat	7.80	6.24	4.68	3.12	1.56	
prec	6.20	4.96	3.72	2.48	1.24	
resl	7.07	5.65	4.24	2.83	1.41	
team	5.48	4.38	3.29	2.19	1.01	
Effort multipliers:						
acap	1.42	1.19	1.00	0.85	0.71	
aexp	1.22	1.10	1.00	0.88	0.81	
cplx	0.73	0.87	1.00	1.17	1.34	1.74
data		0.90	1.00	1.14	1.28	
docu	0.81	0.91	1.00	1.11	1.23	
ltex	1.20	1.09	1.00	0.91	0.84	
pcap	1.34	1.15	1.00	0.88	0.76	
pcon	1.29	1.12	1.00	0.90	0.81	
plex	1.19	1.09	1.00	0.91	0.85	
pvol		0.87	1.00	1.15	1.30	
rely	0.82	0.92	1.00	1.10	1.26	
ruse		0.95	1.00	1.07	1.15	1.24
sced	1.43	1.14	1.00	1.00	1.00	
site	1.22	1.09	1.00	0.93	0.86	0.80
stor			1.00	1.05	1.17	1.46
time			1.00	1.11	1.29	1.63
tool	1.17	1.09	1.00	0.90	0.78	

Fig. 9 COCOMO: co-efficients

	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. 10 THREAT: an example threat table

The details of the THREAT calculations are shown in Figure 11. The threat tables of the current model are shown in Figure 12.

4.3 COQUALMO: defect introduction and removal

COQUALMO models how process options *add* and *remove* defects to software during *requirements*, *design*, and *coding*. For example, poor documentation leads to more errors since developers lack the guidance required to code the right system. So, Figure 13 offers its large defect introduction values when the effort multiplier `docu=vl` is very low. See also Figure 14 for the defects introduced by various settings to the scale factors.

As shown in Figure 15 the COQUALMO defect introduction factors are effects-per-1000 lines of code. A small weighting factor (10,20,30) is added to show an increasing number of defects as the life cycle progresses.

The defects remaining in software is the product of the defects introduced times the percentage removed (see Figure 16 and Figure 17). The removal percentage is calculated in Figure 18 which shows how various actions (`automated analysis`, `peer reviews`, and `execution testing and tools`) remove defects during *requirements*, *design* and *coding*. These values are ratios per 1000 lines of code so their complement represents the remaining defects (see Figure 18).

```
Total_threat =
 (Schedule_threat  + Product_threat   +
  Personnel_threat + Process_threat   +
  Platform_threat  + Reuse_threat)/3.73


Schedule_threat=
  Sced_Rely_threat + Sced_Time_threat +
  Sced_Pvol_threat + Sced_Tool_threat +
  Sced_Acap_threat + Sced_Aexp_threat +
  Sced_Pcap_threat + Sced_Plex_threat +
  Sced_Ltex_threat +
  Sced_Pmat_threat


Product_threat =
  Rely_Acap_threat + Rely_Pcap_threat +
  Cplx_Acap_threat + Cplx_Pcap_threat +
  Cplx_Tool_threat + Rely_Pmat_threat +
  Sced_Cplx_threat + Sced_Rely_threat +
  Sced_Time_threat + Ruse_Aexp_threat +
  Ruse_Ltex_threat


Personnel_threat =
  Pmat_Acap_threat + Stor_Acap_threat +
  Time_Acap_threat + Tool_Acap_threat +
  Tool_Pcap_threat + Ruse_Aexp_threat +
  Ruse_Ltex_threat + Pmat_Pcap_threat +
  Stor_Pcap_threat + Time_Pcap_threat +
  Ltex_Pcap_threat + Pvol_Plex_threat +
  Sced_Acap_threat + Sced_Aexp_threat +
  Sced_Pcap_threat + Sced_Plex_threat +
  Sced_Ltex_threat + Rely_Acap_threat +
  Rely_Pcap_threat + Cplx_Acap_threat +
  Cplx_Pcap_threat + Team_Aexp_threat


Process_threat =
  Tool_Pmat_threat + Time_Tool_threat +
  Tool_Pmat_threat + Team_Aexp_threat +
  Team_Sced_threat + Team_Site_threat +
  Sced_Tool_threat + Sced_Pmat_threat +
  Cplx_Tool_threat + Pmat_Acap_threat +
  Tool_Acap_threat + Tool_Pcap_threat +
  Pmat_Pcap_threat


Platform_threat =
  Sced_Time_threat + Sced_Pvol_threat +
  Stor_Acap_threat + Time_Acap_threat +
  Stor_Pcap_threat + Pvol_Plex_threat +
  Time_Tool_threat


Reuse_threat =
  Ruse_Aexp_threat + Ruse_Ltex_threat
```

Fig. 11 THREAT: the calculations.

5 Handling the Output: Learning

The output from the above learners are passe to the BORE pre-processor, then to the TAR3 treatment learner.

5.1 Multi-Dimensional Optimization using "BORE"

Our goal is reducing development effort *and* the risk of schedule risk *and* the defect density in our code. Optimizing for all these three goals can be difficult. The last 3 columns of Figure 19 show some scores from COCOMO, the risk model, and COQUALMO. The rows are sorted by the COQUALMO scores; i.e. by the estimated number of defects per 1000 lines

		vl	l	n	h	vh	xh
	rely						
sced	vl				1	2	
	l					1	
	cplx						
sced	vl				1	2	4
	l					1	2
	n						1
	time						
sced	vl				1	2	4
	l					1	2
	n						1
	pvol						
sced	vl				1	2	
	l					1	
	tool						
sced	vl	2	1				
	l	1					
	pexp						
sced	vl	4	2	1			
	l	2	1				
	n	1					
	pcap						
sced	vl	4	2	1			
	l	2	1				
	n	1					
	aexp						
sced	vl	4	2	1			
	l	2	1				
	n	1					
	acap						
sced	vl	4	2	1			
	l	2	1				
	n	1					
	ltex						
sced	vl	2	1				
	l	1					
	pmat						
sced	vl	2	1				
	l	1					

		vl	l	n
	acap			
rely	n	1		
	h	2	1	
	vh	4	2	1
	pcap			
rely	n	1		
	h	2	1	
	vh	4	2	1
	acap			
cplx	h	1		
	vh	2	1	
	xh	4	2	1
	pcap			
cplx	h	1		
	vh	2	1	
	xh	4	2	1
	tool			
cplx	h	1		
	vh	2	1	
	xh	4	2	1
	pmat			
rely	n	1		
	h	2	1	
	vh	4	2	1
	acap			
pmat	vl	2	1	
	l	1		
	acap			
stor	h	1		
	vh	2	1	
	xh	4	2	1
	acap			
time	h	1		
	vh	2	1	
	xh	4	2	1
	acap			
tool	vl	2	1	
	l	1		
	pcap			
tool	vl	2	1	
	l	1		

		vl	l	n
	aexp			
ruse	h	1		
	vh	2	1	
	xh	4	2	1
	ltex			
ruse	h	1		
	vh	2	1	
	xh	4	2	1
	pcap			
pmat	vl	2	1	
	l	1		
	pcap			
stor	h	1		
	vh	2	1	
	xh	4	2	1
	pcap			
time	h	1		
	vh	2	1	
	xh	4	2	1
	pcap			
ltex	vl	4	2	1
	l	2	1	
	n	1		
	pexp			
pvol	h	1		
	vh	2	1	
	pmat			
tool	vl	2	1	
	l	1		
	tool			
time	vh	1		
	xh	2	1	
	aexp			
team	vl	2	1	
	l	1		
	sced			
team	vl	2	1	
	l	1		
	site			
team	vl	2	1	
	l	1		

Fig. 12 THREAT: the details. For example, looking at the top-left matrix, the `Sced_Rely_risk` is highest when the reliability is very high but the schedule pressure is very tight.

	rely	data	ruse	docu	cplx	time	stor	pvol	acap	pcap	pcon	aexp	plex	ltex	tool	site	sced
requirements:																	
xh			1.05		1.32	1.08	1.08	1.16								0.83	
vh	0.7	1.07	1.03	0.86	1.21	1.05	1.05	1.1	0.75	1	0.82	0.81	0.9	0.93	0.92	0.89	0.85
h	0.85	1.04	1.02	0.93	1.1	1.03	1.03	1.05	0.87	1	0.91	0.91	0.95	0.97	0.96	0.95	0.92
n	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
l	1.22	0.93	0.95	1.08	0.88			0.86	1.17	1	1.11	1.12	1.05	1.04	1.05	1.1	1.09
vl	1.43			1.16	0.76				1.33	1	1.22	1.24	1.11	1.07	1.09	1.2	1.18
design:																	
xh			1.02		1.41	1.2	1.18	1.2								0.83	
vh	0.69	1.1	1.01	0.85	1.27	1.13	1.12	1.13	0.83	0.85	0.8	0.82	0.86	0.88	0.91	0.89	0.84
h	0.85	1.05	1	0.93	1.13	1.06	1.06	1.06	0.91	0.93	0.9	0.91	0.93	0.91	0.96	0.95	0.92
n	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
l	1.23	0.91	0.98	1.09	0.86			0.83	1.1	1.09	1.13	1.11	1.09	1.07	1.05	1.1	1.1
vl	1.45			1.18	0.71				1.2	1.17	1.25	1.22	1.17	1.13	1.1	1.2	1.19
coding:																	
xh			1.02		1.41	1.2	1.15	1.22								0.85	
vh	0.69	1.1	1.01	0.85	1.27	1.13	1.1	1.15	0.9	0.76	0.77	0.88	0.86	0.82	0.8	0.9	0.84
h	0.85	1.05	1	0.92	1.13	1.06	1.05	1.08	0.95	0.88	0.88	0.94	0.94	0.91	0.9	0.95	0.92
n	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
l	1.23	0.91	0.98	1.09	0.86			0.82	1.05	1.16	1.15	1.07	1.08	1.11	1.13	1.09	1.1
vl	1.45			1.18	0.71				1.11	1.32	1.3	1.13	1.16	1.22	1.25	1.18	1.19

Fig. 13 COQUALMO: effort multipliers and defect introduction

	prec	flex	resl	team	pmat
requirements:					
xh	0.7	1	0.76	0.75	0.73
vh	0.84	1	0.87	0.87	0.85
h	0.92	1	0.94	0.94	0.93
n	1	1	1	1	1
l	1.22	1	1.16	1.17	1.19
vl	1.43	1	1.32	1.34	1.38
design:					
xh	0.75	1	0.7	0.8	0.61
vh	0.87	1	0.84	0.9	0.78
h	0.94	1	0.92	0.95	0.89
n	1	1	1	1	1
l	1.17	1	1.22	1.13	1.33
vl	1.34	1	1.43	1.26	1.65
coding:					
xh	0.81	1	0.71	0.86	0.63
vh	0.9	1	0.84	0.92	0.79
h	0.95	1	0.92	0.96	0.9
n	1	1	1	1	1
l	1.12	1	1.21	1.09	1.3
vl	1.24	1	1.41	1.18	1.58

Fig. 14 COQUALMO: scale factors and defect introduction

```
function defectsIntroduced() {
 return 10*Ksloc()*defectsIntroduced1("requirements") +
        20*Ksloc()*defectsIntroduced1("design") +
        30*Ksloc()*defectsIntroduced1("coding") }

function defectsIntroduced1(table) {
  # return the product of the Figure 13 and
  # and the Figure 14 figures  }
```

Fig. 15 COQUALMO: defects introduced.

	automated analysis	peer reviews	execution_testing _and_tools
requirements:			
xh	0.4	0.7	0.6
vh	0.34	0.58	0.57
h	0.27	0.5	0.5
n	0.1	0.4	0.4
l	0	0.25	0.23
vl	0	0	0
design:			
xh	0.5	0.78	0.7
vh	0.44	0.7	0.65
h	0.28	0.54	0.54
n	0.13	0.4	0.43
l	0	0.28	0.23
vl	0	0	0
coding:			
xh	0.55	0.83	0.88
vh	0.48	0.73	0.78
h	0.3	0.6	0.69
n	0.2	0.48	0.58
l	0.1	0.3	0.38
vl	0	0	0

Fig. 16 COQUALMO: defect removal

```
function Total_defects() {
  return defects("requirements",Coqualr) +
         defects("design",      Coquald) +
         defects("coding",      Coqualc)
}

function defects(what,table) {
  introduced     = defectsIntroduced1(what,table);
  percentRemoved = defectsRemovedRatio(what);
  return percentRemoved*introduced
}
```

Fig. 17 COQUALMO: defects added and removed

```
function defectsRemovedRatio(table, auto,review,tool) {
  return (1 - drf(table,"automated_analysis")) *
         (1 - drf(table,"peer_reviews")) *
         (1 - drf(table,"execution_testing_and_tools"))
}

function drf(table,x ) {
  # return x's value in table from Figure 16
}
```

Fig. 18 COQUALMO: ratio of defects removed

26 inputs							*3 outputs*		
rely	*plex*	*ksloc*	…	*pcap*	*time*	*aa*	*effort*	*schedule threats*	*defects*
5	*1*	*118.80*	…	*5*	*3*	*5*	*2083*	*69*	*0.50*
5	*1*	*105.51*	…	*1*	*3*	*5*	*4441*	*326*	*0.86*
5	*4*	*89.26*	…	*3*	*5*	*3*	*1242*	*63*	*0.96*
5	*2*	*89.66*	…	*1*	*4*	*5*	*2118*	*133*	*2.30*
5	*1*	*105.45*	…	*2*	*4*	*5*	*6362*	*170*	*2.66*
5	*3*	*118.43*	…	*2*	*6*	*2*	*7813*	*112*	*4.85*
5	*4*	*110.84*	…	*4*	*4*	*4*	*4449*	*112*	*6.81*
…									

Fig. 19 Some sample XOMO output.

of code. Interestingly, high number of remaining defects are not correlated with high schedule risk or development effort:

– The second and last rows have *similar* efforts but very *different* defect densities.
– Row two has the *highest* schedule threat but one of the *lowest* defect densities.

The reason for these non-correlations is simple: even though the three models within XOMO using the *same* variables, they predict for *different* goals. This complicates optimization since any gain achieved in one dimension may have detrimental effects on other dimensions.

To model this multi-dimensional optimization problem, XOMO uses a multi-dimensional classification scheme called BORE (short for "best or rest"). BORE maps simulator outputs into a hypercube which has one dimension for each utility; in our case, one dimension for effort, remaining defects, and schedule risk, These utilities are normalized to "zero" for "worst", and "one" for "best". The corner of the hypercube at 1,1,... is the *apex* of the cube and represents the desired goal for the system. All the examples are scored by their normalized Euclidean distance to the apex.

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

	rely	*plex*	*ksloc*	…	*pcap*	*time*	*aa*	*effort*	*secdRisk*	*defects*
best:										
	5	*4*	*89.26*	…	*3*	*5*	*3*	*1242*	*63*	*0.96*
	5	*1*	*118.80*	…	*5*	*3*	*5*	*2083*	*69*	*0.50*
	5	*2*	*89.66*	…	*1*	*4*	*5*	*2118*	*133*	*2.30*
rest:										
	5	*1*	*105.51*	…	*1*	*3*	*5*	*4441*	*326*	*0.86*
	5	*4*	*110.84*	…	*4*	*4*	*4*	*4449*	*112*	*6.81*
	5	*3*	*118.43*	…	*2*	*6*	*2*	*7813*	*112*	*4.85*

Fig. 20 BORE: classification of Figure 19.

outlook	*temp(°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. 21 TAR3: Playing golf.

($Z_i = 0$) 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". For example, BORE might divide Figure 19 into Figure 20.

5.2 Treatment Learning with TAR3

Once the above models run, and BORE classifies the output into $best$ and $rest$, a data miner is used to find input settings that select for the better outputs. This study uses the TAR3 data miner since this learning method return the *smallest* theories that *most* effect the output. In terms of software process changes, such minimal theories are useful since they require the fewest management actions to improve a project.

TAR3 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. $\{U_1 < U_2 < .. < U_C\}$. With E, these classes occur at frequencies $F_1\%, F_2\%, ..., F_C\%$. 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 $f_1\%, f_2\%, ...f_C\%$. TAR3 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 21. In that log, we only play *lots* of golf in $\frac{6}{5+3+6} = 43\%$ 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 22. 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.

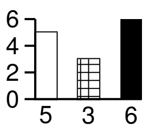
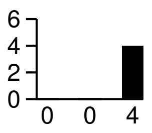
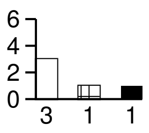
The tests in the `WHERE` clause of the select statements in Figure 22 is a treatment. Classes in treatment learning get a score U_C and the learner uses this to assess the class frequencies resulting from *applying a treatment* (i.e. using them in a `WHERE` clause). In normal operation, a treatment learner 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 [12] that uses stochastic search to find its treatments.

5.3 Iterative TAR3

Sometimes, one round of TAR3 is not enough. *Iterative TAR3* runs by conducting multiple Monte Carlo simulations over

input:	`SELECT class FROM golf`	`SELECT class FROM golf WHERE outlook = 'overcast'`	`SELECT class FROM golf WHERE humidity >= 90`
output:	`none none none none none some some some lots lots lots lots lots lots`	`lots lots lots lots`	`none none none some lots`
distributions:			

legend: ▢ none ▦ some ■ lots

Fig. 22 TAR3: Class distributions selected by different conditions in Figure 21.

the ranges of any uncertain variables. For example, there are 28 variables in the XOMO models:

- Ksloc;
- 5 scale factors;
- 17 effort multipliers;
- 2 calibration parameters ("`a,b`");
- 3 defect removal activities (automated analysis, peer reviews, execution testing and tools).

In the case studies below, only a partial description of some of these variables are available. Hence, learning is a process of sampling from the known constraints, finding the best treatment, then revising the constraints. After, say, 1000 Monte Carlo runs, BORE classifies the outputs as (say) either the 333 *best* or 667 *rest*. The treatment learner studies the results and notes which input ranges select for *best*. The ranges found by the learner then become restraints for future simulations. The whole cycle looks like this:

$$restraints_i \rightarrow simulation_i \rightarrow learn \rightarrow$$
$$\rightarrow restraints_{i+i} \rightarrow simulation_{i+1}$$

6 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 5;
`-p $ares.dat`: load the ARES ranges Figure 6;
`-P ares`: focus the simulation on ARES;
`-s '<$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
```

For example, Figure 23 shows a typical *xomo.out* file and Figure 24 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 24;
2. *xomo.names* : a data dictionary for the data file, shown in Figure 25;
3. *xomo.cfg* : some control settings, shown in Figure 26.

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 25, _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

=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. 23 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. 24 BORE output. In the last column, _0 denotes "rest" and _1 denotes "best".

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).
– 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 26 could be improved but, as we shall see, they suffice for the XOMO data.

7 Case Studies

7.1 Case Study 1

Figure 27 shows the effects of applying Figure 1 to the GNC model of ARES1 modeling in Figure 6. After each round, TAR3 learns treatments and the best treatment is selected to restrain the Monte Carlo simulation of the next round. At

```
_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. 25 xomo.names.

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

Fig. 26 xomo.cfg

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 6. 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.

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.

One prior XOMO-based analysis of the trade space within the space shuttle sub-system offered far more dramatic results than Figure 27 [20]. 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.

round1:
sced = 3

round2:
execution_testing_and_tools = 6

round3:
peer_reviews = 6

round4:
automated_analysis = 6

round5:
ltex = 5

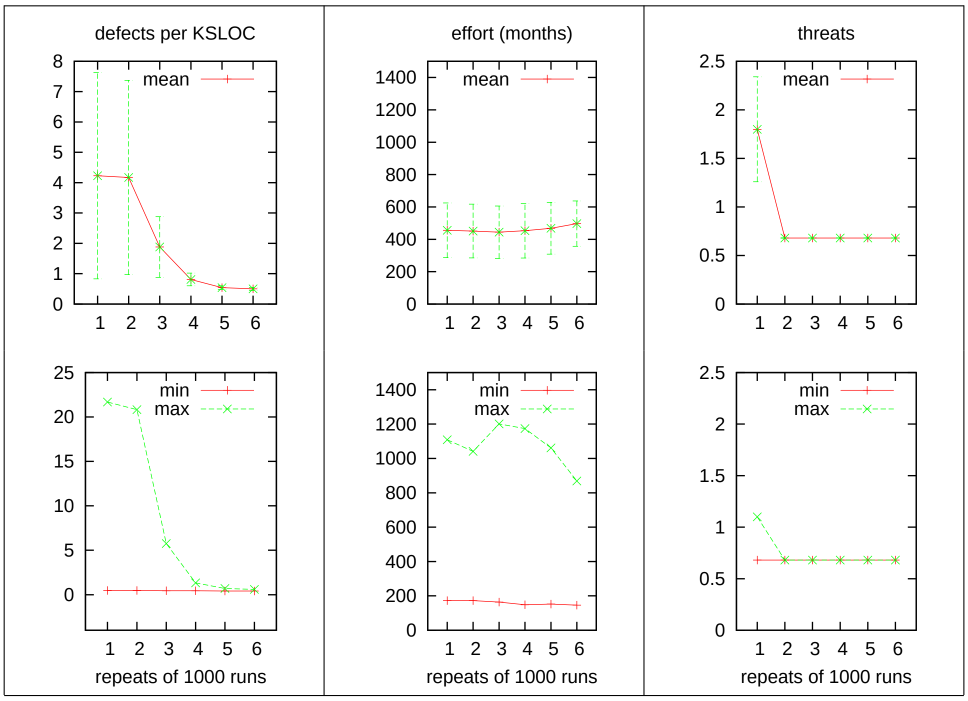


Fig. 27 Results. In the top row, the green error bars denote ± one standard deviation about the mean.

						defects			
round	sced	peer reviews	ltex	execution testing and tools	automated analysis	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. 28 Effects of increasing automated analysis.

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.

7.1.1 Discussion: Case Study 1 The results of Figure 27 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.
– 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 28 and Figure 29 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 28 and Figure 29 show the effects of only changing the level of

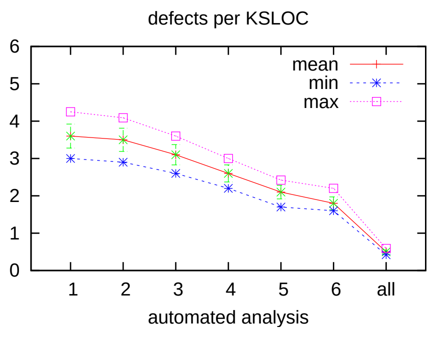


Fig. 29 Figure 28, plotted. Green error bars denote ± one standard deviation about the mean.

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 29 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.

7.2 Case Study 2

This second study explores a less constrained problem than the first study. Figure 30 shows the input ranges for software from "KC1"- an experimental NASA space plane. KC1 was the seed of the design for the Orbital Space Plane which, in turn, was used for the initial design work for ARES. Note that the slots can vary over a wider range of values.

Figure 31 shows the effects of iterative treatment learning with TAR3 on KC1. In the first round of learning, chang-

```
system with kc1
    A = 2.94
        B = 0.91
acap just 2 3
aexp just 2 3
cplx  just 5 6
data = 3
docu just 2 4
ksloc just  75 125
ltex just 2 4
pcap = 3
pcon just 2 3
plex = 3
pmat just 1 4
prec just 1 2
pvol = 2
rely = 5
resl just 1 3
ruse  just 2 4
sced just 1 3
site = 3
stor just 3 5
team just 2 3
tool just 2 3
    flex just 2 5
```

Fig. 30 The ranges of study 2.

round1:
time just 3 4 and pmat just 3 4

round2:
acap=3 and sced=3

round3:
flex just 4 5 and ltex = 4

round4:
ksloc just 75 91 and pmat = 4 and tool = 3 and time just 3 4
acap =3

round5:
stor just 3 4 and prec = 2 and aexp = 3

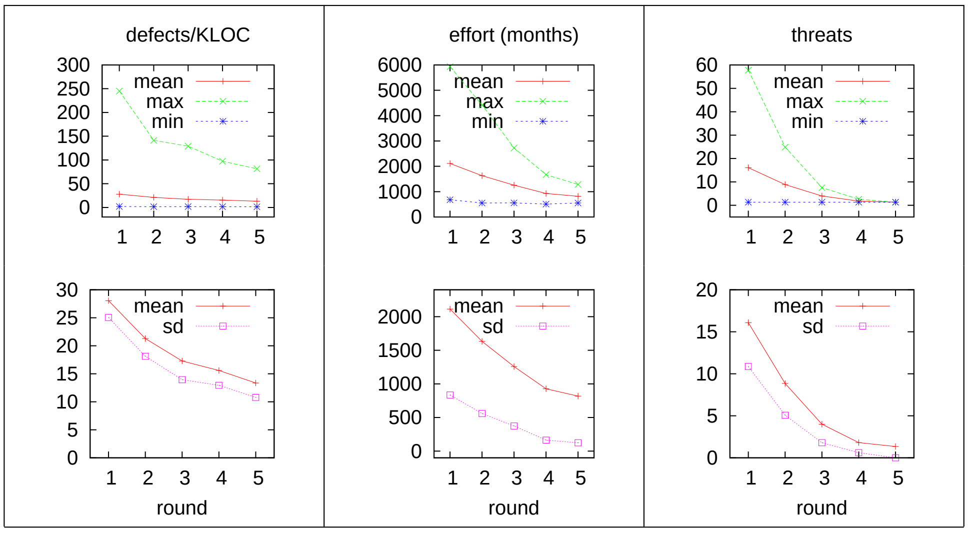


Fig. 31 Results of case study 2.

ing the runtime pressure ($time$) and process maturity ($pmat$) results in moderate reductions in effort/defects/ and threats. Much larger reductions were achieved by round 3 after constraining analyst capability, ($acap$) schedule pressure ($sced$), development flexibility ($flex$) and language and tool experience ($ltex$).

Further rounds offered some further improvements in defects/effort/ and threats but required much more constraints. By round 3, management action would be required to make 6 process changes. Rounds 4 and 5 required 8 more. Hence, in terms of cost/benefit, there might be case to pause after round 3.

7.2.1 Discussion: Case Study 2 The most important feature of Figure 31 is what is *missing*[3]. Note that all the recommendations proposed by the treatment learner were *process*-based, not *product*-based. Missing in the recommendations are any mention of (e.g.) defect removal tools such as the automated analysis or the execution/testing tools discussed in the last case study.

One explanation for these missing features can be found in the different *magnitude* of the problems facing KC1 or ARES. The y-axis of the Figure 31 graphs are much larger than the Figure 27 plots; for example, the largest predicted number of defects faced by ARES and KC1 are 8/KSLOC and 300/KSLOC respectively.

One lesson from these two case studies is that process issues trump product issues. Debating which formal method to apply is irrelevant if your analysts are all low quality. When faced with large defect problems (e.g. KC1's 300/KSLOC), it may well be more important to make process changes rather than hope that applying some tool will solve your project's problems.

[3] "Is there any point to which you would wish to draw my attention?"
"To the curious incident of the dog in the night time."
"The dog did nothing in the night time."
"That was the curious incident," remarked Sherlock Holmes.
–Sir Arthur Conan Doyle. *Silver Blaze*

8 Conclusion

The trade space for NASA systems can be very large. Software models like COCOMO, COQUALMO, and THREAT contain many assumptions about their domain. The conclusions gained from this models should be scrutinized by domain experts. Early in the life cycle of a software project, such scrutiny is complicated by all the unknowns associated with a project. Exploring all those unknowns can lead to massive data overload as domain experts are buried beneath a mountain of data coming from their simulators.

Here, we have discussed tools that automate exploring that space. These tools integrate to other NASA-funded tools via slot trees. XOMO, BORE, and treatment learners like TAR3 can assist in trade space exploration. These tools can find automatically find software process decisions that reduce defects and effort and risk of schedule over run. These tools sample the space of options and report sample conclusions within the space of possibilities.

To demonstrate that technique, this paper conducted case studies with software development for NASA systems. The case studies proved to be very different: one required product patches (ARES) while the other required massive changes to process structure (KC1).

While the particular case studies examined here is quite specific, the analysis method is quite general. There is nothing stopping an analyst from using XOMO to study other kinds of software development. The only requirement is that the essential features of that software can be mapped onto COCOMO-like models.

– All the models used here contain most of their knowledge in easy-to-modify tables representing the particulars of different domains.

– All the tools used here are portable and use simple command-line switches that allow an analyst to quickly run through a similar study for a different kind of project.

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. *Software Engineering Economics*. Prentice Hall, 1981.
3. 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.
4. 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`.
5. 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`.
6. 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`.
7. 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`.
8. N. E. Fenton and M. Neil. Software metrics: A roadmap. In A. Finkelstein, editor, *Software metrics: A roadmap*. ACM Press, New York, 2000. Available from `http://citeseer.nj.nec.com/fenton00software.html`.
9. N. E. Fenton and S. Pfleeger. *Software Metrics: A Rigorous & Practical Approach*. International Thompson Press, 1997.
10. M. Fisher and T. Menzies. Learning iv&v strategies. In *HICSS'06*, 2006. Available from `http://menzies.us/pdf/06hicss.pdf`.
11. 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`.
12. Y. Hu. Treatment learning, 2002. Masters thesis, Unviersity of British Columbia, Department of Electrical and Computer Engineering. In preperation.
13. R. Jensen. An improved macrolevel software development resource estimation model. In *5th ISPA Conference*, pages 88–92, April 1983.
14. R. Madachy. Heuristic risk assessment using cost factors. *IEEE Software*, 14(3):51–59, May 1997.
15. T. Menzies, E. Chiang, M. Feather, Y. Hu, and J. Kiper. Condensing uncertainty via incremental treatment learning. In T. M. Khoshgoftaar, editor, *Software Engineering with Computational Intelligence*. Kluwer, 2003. Available from `http://menzies.us/pdf/02itar2.pdf`.
16. 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`.
17. T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from `http://menzies.us/pdf/03tar2.pdf`.
18. T. Menzies and J. Kiper. Better reasoning about software engineering activities. In *ASE-2001*, 2001. Available from `http://menzies.us/pdf/01ase.pdf`.
19. 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`.
20. 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`. For more details, see also the longer technical report `http://menzies.us/pdf/05xomo101.pdf`.
21. T. Menzies and J. Richardson. Making sense of requirements, sooner. *IEEE Computer*, October 2006. Available from `http://menzies.us/pdf/06qrre.pdf`.
22. T. Menzies, S. Setamanit, and D. Raffo. Data mining from process models. In *PROSIM 2004*, 2004. Available from `http://menzies.us/pdf/04dmpm.pdf`.
23. 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`.
24. 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`.
25. R. Park. The central equations of the price software cost model. In *4th COCOMO Users Group Meeting*, November 1988.
26. L. Putnam and W. Myers. *Measures for Excellence*. Yourdon Press Computing Series, 1992.
27. 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`.