XOMO: Understanding Development Options for Autonomy

Tim Menzies*
Computer Science,
Portland State University
tim@timmenzies.net

Julian Richardson
RIACS/USRA,
NASA Ames Research Center
julianr@email.arc.nasa.gov

Abstract

Autonomy is a key capability for future space exploration, but uncertain risks in the development and deployment of autonomy systems create a barrier to its use.

Our broad goal is to remove this barrier by developing a model which can quantify autonomy risk, and developing tools and techniques which analyze this and other models, e.g. COCOMO-II, to determine actions which can be taken during project development to improve project outcome, for example reducing cost and/or risk.

In this paper, we focus on the second of these — the analysis of models to determine beneficial project actions. We demonstrate that the trade space of project development options can be analyzed by combining XOMO *— a general framework we have developed for Monte Carlo simulation of COCOMO-like models — with data mining (in particular,* treatment learning*), to determine those actions which most improve project outcome. In this paper, we use XOMO to simulate development options and measure their effects according to three models: the COCOMO effort estimation model, the COQUALMO defect model, and Ray Madachy's schedule risk model.*

In a sample case study, the combination of XOMO and treatment learning finds process options which halve the mean development effort, halve the mean risk of schedule over run, reduce the mean defect density by 85%, and significantly reduce the variance on the above measures.

1 Introduction

Autonomy is a key capability for future space exploration, but the risks and many unknowns in the development and deployment of autonomy systems create a significant barrier to its use.

Our broad goal is to remove this barrier by developing a model which can quantify autonomy risk, and developing tools and techniques which analyze this and other models, e.g. COCOMO-II, to determine actions which can be taken during project development to improve project outcome, for example reducing cost and/or risk.

In this paper, we focus on the second of these — the analysis of models to determine beneficial project actions.

At any stage in the development of a project, many aspects of the project are fixed, but there are options for changing some aspects. We show that the trade space of project development options can be analyzed by combining *XOMO*[1] — a general framework we have developed for Monte Carlo simulation of COCOMO-like models — with data mining (in particular, *treatment learning*, to determine those actions which most improve project outcome. In this paper, we use XOMO to simulate development options and measure their effects according to three models:

- The COCOMO effort estimation model [3, p29-57];
- The COQUALMO defect model [3, p254-268];
- The Madachy schedule risk model [3, 284-291].

The data miner used here is the TAR3 *treatment learner* [12]. Treatment learning assumes that we are all busy people and busy people 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 autonomous system), 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.

Our approach can analyze large spaces of options for projects at various stages in the development life cycle. In the late stages of project development, we expect our models to have sufficient fidelity to recommend process improvements, especially during verification and validation, to improve product quality. In the very early stages of development, we believe that our tools are sufficiently fast that

*To appear, 20th International Forum on COCOMO and Software Cost Modeling, October 25-28, 2005, Los Angeles, California. Available online at http://timmenzies.net/pdf/05xomo.pdf.

[1]Download XOMO from hhtp://unbox.org/src/bash/xomo. For help, see http://promise.unbox.org/DataXomo.

they can analyze large trade spaces and assist in selection of mission architecture and design.

In a sample case study, XOMO and treatment achieve all of the following:

1. The mean development effort will be nearly halved;
2. The mean risk of schedule over run will be halved;
3. The mean defects densities will be reduced by 85%.
4. The variance on the above measures will also be significantly reduced.

The case study was fast to run: all the results shown below took ten minutes to run on a standard computer (a Mac OS X box running at 1.5GHz). Those ten minutes included 5000 runs of the model and five runs of the data miners. Hence, this study gives us confidence that AI-based decision support agents can run fast enough to keep up with humans debating software process options for autonomous systems.

The rest of this paper describes how the XOMO models and treatment learning were applied to the case study.

2 Autonomy Risk and Mitigations

The term 'autonomous' denotes different things to different people. In common usage, 'autonomous' systems are those which can operate — at least partly — without human interaction. In NASA's manned space program, the term has a narrower usage, referring to systems which can operate — at least partly — without interaction with ground control. Systems which operate without any human operation are termed automatic. For unmanned systems, the meanings of the two terms coincide.

Autonomy has high potential impact for both manned and unmanned space exploration: autonomy has a number of benefits, and enables key capabilities such as automated rendezvous and docking and integrated system health management (ISHM). Development and use of autonomous systems also entails risks. Especially in the context of manned space exploration, perception of these risks is a barrier to adoption of significant autonomy.

We believe that a tool which could identify and quantify autonomy risks and mitigations would be assist control and understanding of autonomy risks, and improve uptake of autonomy technologies.

We are currently engaged in a NASA-funded project to construct a model of risks and mitigations for autonomy. The main components of the model:

- assist identification of risks and risk mitigations for autonomy projects using detailed taxonomies of risks and risk mitigations,
- *quantify*, using a COCOMO-like sub-model, risks and mitigations which the model user has identified
- recommend risk mitigation strategies based on the above computations.

The main part of this paper (§3 onwards) describes our work on analyzing COCOMO-like models using XOMO. In the next section, we set the scene, and provide a glimpse at part of our risk taxonomy, by outlining some of the benefits and risks of autonomy.

2.1 Benefits and Risks of Autonomy

Autonomy can increase capability, reliability, safety, and reduce cost in many ways, including:

- mitigation of communications limitations:
  - autonomy by definition reduces the reliance on communication with ground control. This directly reduces mission risk — a JPL study [9] of critical errors in unmanned flight concluded that 41% of software anomalies were triggered by communications uplink or downlink problems
  - communications channels have limited bandwidth and inherent delays in data delivery, including light-time delays. Consequently, onboard calculations can benefit from more accurate and timely spacecraft state information. Onboard trajectory calculations can also reduce fuel [8].
- Mitigation of human limitations:
  - reduction in human error
  - processing much larger amounts of data than can be handled by humans
  - reduction in amount of ground support necessary, which may entail cost savings or allow ground operators to carry out other tasks
- Enable new capabilities:
  - increased science by reducing the need for teleoperation, e.g. for autonomous rover operations including single-cycle instrument placement and 'long traverse' autonomous navigation and motion planning
  - automated rendezvous and docking e.g. for on-orbit assembly

A good example of the beneficial use of autonomy is the Deep Impact mission [1], which hit comet Tempel 1 with a 450kg impactor on 4th July 2005. The high relative speeds of the probe and the comet (over 10km/s), and trajectory uncertainties due to the comet's coma and other effects, meant that a pre-calculated trajectory could not be guaranteed to hit the comet. Due to light time delays, ground control could not determine last-minute trajectory corrections, nor correct faults that might occur in the hostile environment

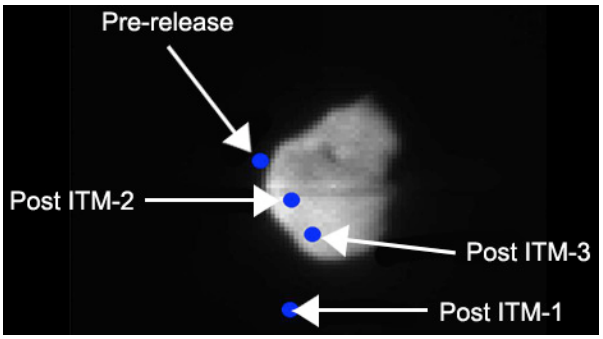


Figure 1. Deep Impact autonomous trajectory corrections. Image credit: NASA/JPL-Caltech/UMD.

close to the comet. Autonomous trajectory corrections and fault recovery allowed the mission to succeed.

In any potential use of autonomy, we have to weigh the benefits of autonomy against potential risks, which can mostly be present in the development and application of any software system but may tend to be worse for autonomous systems, for example:

- Requirements-related risks:
  - Inconsistency, incompleteness
  - Traceability — particularly for autonomous systems, it can be difficult to link high level requirements which state the desired behavior of the system to low level requirements which are more appropriate for implementation and testing
- Testing related risks:
  - Autonomous systems typically sense and process data from their physical environment, resulting in a very large input space. Obtaining adequate test case covered is therefore difficult.
  - Autonomous systems are frequently concurrent — it is hard to detect concurrency-related errors through testing.
- experience-related risks: techniques used for implementing autonomy have little track record in space applications, and flight software developers are frequently not experienced in developing such software
- Model-related risks:
  - Autonomy systems frequently employ an internal model of the system's environment and possible behaviors. Risks arise when the model may be inaccurate or incorrect. The model is sometimes explicit — for example in a planner as a set of plan schemas — and sometimes not.

```
# thousands of lines of codes
_ANY(ksloc, 2,   10000)

# scale factors: exponential effect on effort
ANYi(prec,  1,   6)
ANYi(flex,  1,   6)
...
# effort multipliers: linear effect on effort
ANYi(rely,  1,   5)
...
# defect removal methods
_ANYi(automated_analysis,          1,  6)
_ANYi(peer_reviews,                1,  6)
_ANYi(execution_testing_and_tools, 1,  6)

# calibration parameters
_ANY(a,     2.25,3.25)
_ANY(b,     0.9, 1.1)

function Prec()
  return scaleFactor("prec",
                     prec())
...
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())}
```

Figure 2. Part of XOMO specification of COCOMO-II.

3 XOMO

Some existing COCOMO-like models, e.g. COCOMO II, can be profitably analyzed in order to improve outcome in autonomy development projects. While our model of autonomy risk and mitigations is currently under construction, we believe it will also be COCOMO-like. We have therefore developed a general framework for analysis of COCOMO-like models; in this way we can develop and retain a significant analysis capability without committing too much to any particular model.

XOMO is a general framework for Monte Carlo simulations that has been customized for processing COCOMO-like models. Figure 2 shows part of the specification in XOMO of COCOMO-II, illustrating the following XOMO features:

- declaration of model inputs and allowed ranges (e.g. `prec`, an integer in the range 1..6),
- the functional relationship (here, `Effort()`) between the model inputs and outputs,
- use of lookup tables to map model inputs to real values like the COCOMO effort multipliers and scale factors.

Internally, XOMO generates code for computing the model outputs efficiently using memoed, possible filtered, functions. Experienced programmers can modify the auto-

generation process by editing XOMO's macro expansion files (which are written in the M4 language).

4 Case Study

XOMO model specifications define allowable ranges for model inputs. When applying XOMO analysis in a development particular project, the user can further restrict the allowable ranges of model input variables to reflect project-specific constraints (e.g. that the particular project subject to analysis must have very good analysts). The command line can set exact values (using the "`-=`" flag) or can define a range from some lower to upper value (using the "`-l`" and "`-u`" flags).

Using this syntax, we can define restrictions on COCOMO parameters corresponding to a fictitious case study:

`-p ksloc -l 75 -u 125` : We assume that some developer from a prior autonomy project has guesstimate that this new project will require 75,000 to 125,000 lines of code.

`-p rely -=5` : At NASA, everything must be have highest reliability. In COCOMO, `rely`'s maximum value is very high; i.e. 5.

`-p prec -= 1` : Since this team has never done this sort of thing before, the precedence (or `prec`) is set to the lowest value.

`-p acap -= 5` : This team is skillful; i.e. has highest analyst capability.

`-p aexp -= 1` : Their experience in this kind of software is non-existence.

`-p cplx -= 6` : The software is very complex.

`-p ltex -= 1` : The team has no experience with the languages and tools used for autonomous systems.

`-p ruse -= 6` : This team, in their enthusiasm, believe that the tools they are building here will be reused by many developers in the future.

5 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 4 show scores from COCOMO, the risk model, and COQUALMO. The rows are sorted by the COQUALMO scores; i.e. by the estimated number of defects

```
runxomo() {
  Scenario="-p ksloc -l 75 -u 125
            -p rely -= 5
            -p prec -= 1
            -p acap -= 5
            -p aexp -= 1
            -p cplx -= 6
            -p ltex -= 1
            -p ruse -= 6"
  xomo $Scenario }
```

Figure 3. XOMO: specifying restraints.

26 inputs								*3 outputs*		
rely	*plex*	*ksloc*	...	*pcap*	*time*	*aa*		*effort*	*schedule risk*	*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*
...										

Figure 4. XOMO: output from Figure 3.

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*

Figure 5. BORE: classification of Figure 4.

per 1000 lines 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 risk 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 Euclidean distance to the apex. The N best examples closest to the apex are then labeled $best$. A random sample of N of the remaining examples are then labeled $rest$. Figure 5 shows a BORE report of the three $best$ and three $rest$ examples from XOMO output. Note how the average efforts, schedule risk, and defects are lower in $best$ than $rest$.

BORE's classifications are passed to a data miner to find what settings select for *best* and avoid the *rest*. Before describing that data mining process, we first describe the COCOMO, COQUALMO and schedule risk models that generated the output columns of Figure 4.

6 Models

This section describes the three models within XOMO:

- Boehm et.al.'s COCOMO-II (2000) model that computes development effort;
- Madachy's heuristic risk model that computes the risk that schedules will over run;
- Boehm et.al.'s COQUALMO model that estimates the number of defects remaining in delivered code;

6.1 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,

	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	

Figure 6. COCOMO: co-efficients

COCOMO-II estimates development effort as a function of the number of adjusted thousand lines of code, KSLOC, and attributes of the development process divided into scale factors $SF_1 ... SF_5$: `flex`, `pmat`, `prec`, `resl`, `team`, and effort multipliers $EM_1 ... EM_17$: `acap`, `aexp`, `cplx`, `data`, `docu`, `ltex`, `pcap`, `pcon`, `plex`, `pvol`, `rely`, `ruse`, `sced`, `site`, `stor`, `time`, `tool`:

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

Figure 2 showed the specification in XOMO of the COCOMO effort equation. Values such as `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 6.

Ideally, software effort-estimation models like COCOMO-II should be tuned to their local domain. Off-the-shelf "untuned" models have been up to 600% inaccurate in their estimates, e.g. [15, p165] and [7]. However, tuned models can be far more accurate. For example, [5] reports a study with a Bayesian tuning algorithm using the COCOMO project database. After Bayesian tuning, a cross-validation study showed that COCOMO-II model produced estimates that are within 30% of the actuals, 69% of the time.

Elsewhere, with Boehm, Chen, Port, Hihn, and Stukes [4, 11, 13, 14] we have explored calibration methods for COCOMO. Here, we take a new approach and ask

		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		

Figure 7. SCED-RISK: 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

Figure 8. COQUALMO: effort multipliers and defect introduction

	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

Figure 9. SCED-RISK: an example risk table

"what conclusions hold across the space of possible tunings"?. Hence we treat the tuning parameters "a" and "b" as random variables (see Figure 2, last two lines).

6.2 SCED-RISK: a Heuristic Risk Model

The Madachy Heuristic Risk model (hereafter SCED-RISK) 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 risk"; 5-15 "medium risk"; 15-50 "high risk"; and 50-100 "very high risk". Studies with the COCOMO-I project database have shown that the Madachy SCED-RISK index correlates well with $\frac{months}{KDSI}$ (where KDSI is thousands of delivered source lines of code) [10].

Internally, the model contains dozens of tables of the form of Figure 9. Each such table adds some "riskiness" 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`). Recalling Figure 6, 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 risk factor of 2 (see Figure 9). Hence, a project with these two attribute ranges would contribute 1.43*1.26*2=3.6036 to the schedule risk.

The risk tables of the current model are shown in Figure 7.

6.3 COQUALMO: defect introduction, 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 8 offers its large defect introduction values when the effort multiplier `docu=vl` is very low. See also Figure 10 for the defects introduced by various settings to the scale factors. For the purpose of our analysis, we combine the estimates of requirements, design and coding defects into a single defect measure using a weighted sum with weights 10, 20, 30 for requirements, design and code defects respectively.

	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

Figure 10. COQUALMO: scale factors and defect introduction

	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

Figure 11. COQUALMO: defect removal

The defects remaining in software is the product of the defects introduced (see Figure 10) times the percentage removed (see Figure 11). These values are ratios per 1000 lines of code.

7 Learning

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 treatment learning 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.

7.1 Treatment Learning

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. $\{U_1 < U_2 < .. < U_C\}$. With E, these classes occur at frequencies $F_1\%, F_2\%, ..., F_C\%$. A treatment T of size N is a conjunction of attribute ranges $\{R_1 \wedge R_2 ... \wedge R_N\}$. 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:

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

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 situations where more of the outcomes are favorable. Treatment learning is different to other contrast set learners like STUCCO [2], 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 [6] that uses stochastic search to find its treatments.

7.2 Iterative Treatment Learning

Sometimes, one round of treatment learning is not enough. *Iterative treatment learning* runs by conducting Monte Carlo simulations over the ranges of any uncertain variables. For example, there are 28 variables in the models analyzed in our case study:

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

The restraints of Figure 3 only offer hard constraints on seven of the variables: `rely, prec, acap, ...`[2]. XOMO's Monte Carlos execute by picking random values from valid ranges for all known inputs. After, say, 1000 Monte Carlo runs, BORE classifies the outputs as either the 100 $best$ or 100 $rest$. 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:

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

8 Results

For this study the initial baseline restraints were set according to our case study from §4.

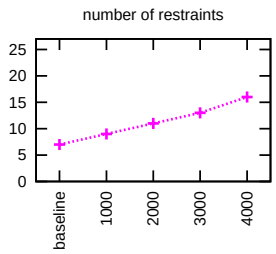
XOMO was run 1000 times each iteration and BORE returned the 100 $best$ examples and a random sample of 100 of the $rest$. These $best$ and $rest$ examples were passed to TAR3 and the best learned treatment was imposed as restraints on subsequent iterations.

Figure 12 shows the restraints learned by four iterations of iterative treatment learning. Figure 13 shows the effects of these restraints on the output of the `XOMO` models:

1. The mean development effort was nearly halved: 3257 to 1780 months;
2. The mean SCED-RISK halved: 77 to 36;
3. The mean defects densities were reduced by 85% from 0.97 to 0.15.
4. The variances on the above measures were significantly reduced: the COQUALMO and SCED-RISK standard deviations nearly reached zero.

Several of the Figure 13 curves flatten out after 2000 runs of `XOMO`. A parsimonious management strategy could be formed from just the results of the first two rounds of learning. Interestingly, in those first two rounds, process changes were more important than the application of technology. Technology-based techniques such as `tool` support or `execution testing and tools` did not

[2]The constraint on `ksloc` is softer- it can vary from 75K to 125K).



baseline	learned restraints			
	1000	2000	3000	4000
$75 \leq$ ksloc ≤ 125	sced=4	pmat=5	tool=4	team=5
rely = 5	peer_reviews=5	pcap=4	execution_testing-	resl=5
prec = 1			_and_tools=5	automated-
acap = 5				analysis=5
aexp = 1				
cplx = 6				
ltex = 1				
ruse = 6				

Figure 12. TAR3: learned restraints

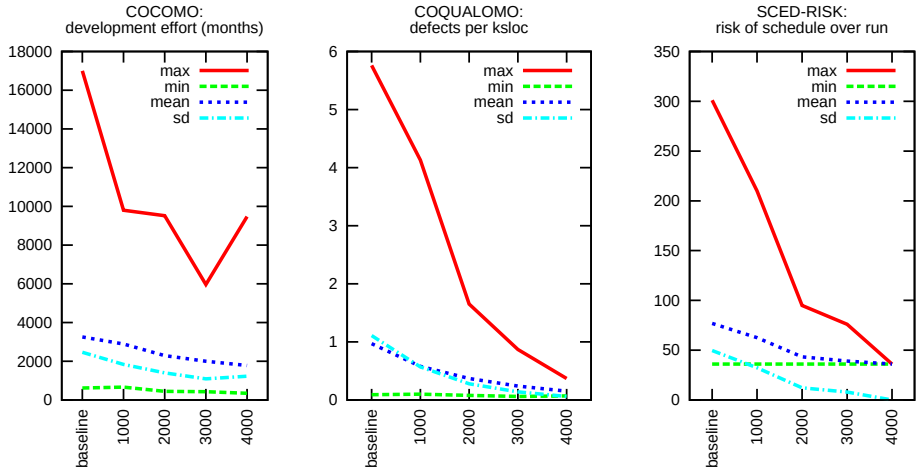


Figure 13. TAR3: impact of Figure 12's restraints.

arise till iteration three. On the other hand, the first two iterations labeled "1000,2000" in Figure 12 want to decrease schedule pressure (`sced`), increase process maturity (`pmat`), and programmer capability (`pcap`) and requested the user of peer reviews.

Another interesting feature of the results is that many of the inputs were never restrained. The left-hand-side plot of Figure 12 shows that even after four rounds of learning, only 17 of the 28 inputs were restrained. That is, management commitments to 11 of the 28 inputs would have been a waste of time. Further, if management is content with the improvements gained from the first two iterations, then only 12 restraints are required and decisions about the remaining 16 inputs would have been superfluous.

9 Discussion

Software models like COCOMO, COQUALMO, and SCED-RISK contain many assumptions about their domain. The conclusions gained from these 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.

Tools like XOMO, BORE, and treatment learners like TAR3 can assist in that scrutiny. 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 a case study with software development for autonomous systems. Certain special features of autonomous systems were identified. These features included high complexity and little experience with building these kinds of systems in the past. These features were then mapped into general software cost and risk models.

It is encouraging that the analysis is so fast: the above case study took less than ten minutes to run on a standard computer. Hence, we can use these tools during early life

cycle debates about options within a software project.

While the particular case study examined here is quite specific, the analysis method is quite general. Our case study related to autonomous systems, but 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.

10 Acknowledgments

This research was conducted for RIACS (an institute of the Universities Space Research Association (USRA)) under a grant from NASA's Exploration Systems Mission Directorate, and at Portland State University under USRA subcontract 8023-004. XOMO is available from the authors 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.

References

[1] M. A'Hearn, A. Delamere, and W. Frazier. The deep impact mission: Opening a new chapter in cometary science. In *Proceedings 51st International Astronautical Congress*, october 2000. IAA-00-IAA.11.2.04.

[2] 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`.

[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] Z. Chen, T. Menzies, and D. Port. Feature subset selection can improves software cost estimation. In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies/pdf/05/fsscocomo.pdf`.

[5] S. Chulani, B. Boehm, and B. Steece. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineerining*, 25(4), July/August 1999.

[6] Y. Hu. Treatment learning, 2002. Masters thesis, Unviersity of British Columbia, Department of Electrical and Computer Engineering. In preperation.

[7] C. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.

[8] S. Lee and A. G. Santo. Tradeoffs in functional allocation between spacecraft autonomy and ground operations: the NEAR (Near Earth Asteroid Rendezvous) experience, August 2004.

[9] R. Lutz. Patterns of software defect data on spacecraft. In *2nd International Conference on Space Mission Challgenges for Information Technology*, 2003.

[10] R. Madachy. Heuristic risk assessment using cost factors. *IEEE Software*, 14(3):51–59, May 1997.

[11] T. Menzies, Z. Chen, D. Port, and J. Hihn. Simple software cost estimation: Safe or unsafe? In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies.us/pdf/05safewhen.pdf`.

[12] T. Menzies and Y. Hu. Data mining for very busy people. In *IEEE Computer*, November 2003. Available from `http://menzies.us/pdf/03tar2.pdf`.

[13] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes. Specialization and extrapolation of induced domain models: Case studies in software effort estimation. 2005. IEEE ASE, 2005, Available from `http://menzies.us/pdf/05learncost.pdf`.

[14] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes. Validation methods for calibrating software effort models. In *Proceedings, ICSE*, 2005. Available from `http://menzies.us/pdf/04coconut.pdf`.

[15] T. Mukhopadhyay, S. Vicinanza, and M. Prietula. Examining the feasibility of a case-based reasoning tool for software effort estimation. *MIS Quarterly*, pages 155–171, June 1992.