Software Cost Models: When Less is More

Zhihao Chen[†], Tim Menzies[⋆], Dan Port[‡], Barry Boehm[†]
[†]Center for Software Engineering, University. of Southern California, USA
[⋆]Computer Science, Portland State University
[‡]Computer Science, University of Hawaii
zhihaoch@cse.usc.edu;tim@timmenzies.net;dport@hawaii.edu;boehm@cse.usc.edu

Abstract

If experience tells us when to *add* variables to a cost model, it should also be able to tell us when to *subtract* variables. Data mining methods can intelligently select variables to subtract and so generate reduced models which are much better at estimating efforts within an organization's particular context.

Word count: 3850 words+ (7 figures*200)=5150

1 Introduction

Good software cost models can significantly help the managers of software projects. With such good models, project stake holders can make informed decisions about (e.g.) "buy-or-make", how to manage resources, how to control and plan the project, and how to deliver the project on time, on schedule and on budget.

As we learn more about software, the number of parameters in those models tends to grow. In going from the 1981 COCOMO model [1] to the 2000 COCOMO II model [2], one parameter, "Turnaround Time", was dropped to reflect the almost-universal use of interactive software development. Another parameter, "Modern Programming Practices", was dropped in favor of a more general "Process Maturity" parameter. But several more parameters were added to reflect the subsequently-experienced influences of such factors as "Development for Reuse", "Multisite Development", "Architecture and Risk Resolution", and "Team Cohesion". The COCOMO II book [2] also provides capabilities and guidelines for an organization to add new parameters, reflecting their particular situations.

			%estimates within 30% of actuals		
	data set	#projects	before	after	$\frac{after}{before}$
1.	nasa60	60	68%	82%	120%
2.	coc81	63	44%	51%	116%
3.	cii04	119	68%	72%	106%
4.	cii00	161	75%	76%	101%
		mean	64%	70%	110%

	data set	#projects	before	after	$\frac{after}{before}$
5.	t03	10	28%	62%	221%
6.	p03	12	8%	52%	650%
7.	c02	12	11%	48%	436%
8.	c01	13	20%	58%	290%
9.	t02	14	19%	81%	426%
10.	p04	14	25%	63%	252%
11.	p02	22	15%	97%	646%
12.	c03	31	88%	95%	108%
		mean	27%	70%	379%

Figure 1: Estimation and variable subtraction.

But if experience can tell us when to *add* variables, it should also be able to tell us when to *subtract* variables. If not, then our models may grow forever and become needlessly complex. Figure 1 shows that subtracting variables can be surprisingly useful. For data sets containing many projects, variable subtraction made little difference. But for data sets containing just a few dozen projects (or less), the improvement in estimation effectiveness was dramatic (on average, a 379% improvement).

[0]Submitted to IEEE Software, May 27, 2005. Earlier drafts available at: http://menzies.us/pdf/05lessismore.pdf.

2 When Less is NOT More

The improvements seen in Figure 1 are quite large and the rest of this paper discusses the data mining methods used to find which variables to subtract. Before that, we first describe situations in which variables should not be subtracted, *even it subtracting them improves estimation effectiveness*.

Firstly, our variable subtraction methods require an historical database of projects. If there is *no* such database,then our variable subtraction techniques won't work.

Secondly, even a historical database exists and our techniques suggest subtracting variable X, then it may still be important to ignore that advice. If a cost model *ignores* certain effects that business users believe are important, then those users may *not trust* that model. In that case, even if a variable has no noticeable impact on predictions, it should be left in the model. By leaving such variables in a model we are acknowledging that, in many domains, expert business users hold in their head more knowledge that what may be available in historical databases. Suppose that there is some rarely occurring combination of factors which leads to a major productivity improvement. Even if there is little data on some situation, it still should be included in the model. For example, even though some studies have shown that reduced-parameter function point counting rules are equally good in most situations [2], COCOMO II supports the full International Function Point Users' Group (IFPUG) set of parameters due to their wide usage and acceptance in the IFPUG community.

Thirdly, another reason not to subtract variables is that you still might need them. For example, the experiments shown below often subtract over half of the attributes in a COCOMO-I model. As shown in Figure 1, this can dramatically improve the effort estimation. However, suppose that a business decision has to be made using some of the subtracted variables. The reduced model has no information on those subtracted variables so a business user would have to resort to other information for making their decisions.

For these reasons, we propose the *decision ladder* of Figure 2. The ladder has three parts: general COCOMO, followed by local calibration, followed by variable reduction. The ladder represents how much variable subtraction is suitable for different business situations.

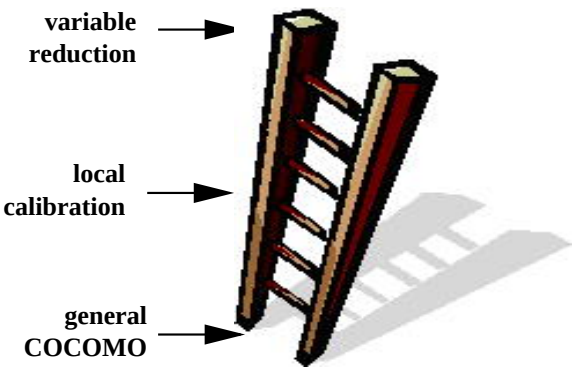

Figure 2: A ladder of decisions in cost modeling.

At the base of the ladder is *general public domain knowledge*. The 1981 regression co-efficients of COCOMO-I or the updated co-efficients of COCOMO-II [2] are the best general-purpose indicators we can currently offer for cost estimation. Management decisions can use that public knowledge to make software process decisions. For example, according to the coefficients on the COCOMO-II *pmat* variable, the increase in cost between a CMM3 and CMM4 project contain N lines of code is $N^{3.13}/N^{1.56}$. With this estimate in hand, a business user could then make their own assessment about the cost of increased software process maturity vs the benefits of that increase.

If historical data from the local site is available, then effort estimation can rise to the next rung in the estimation ladder. COCOMO-I and COCOMO-II contain with several *local calibration* variables that can quickly tune a model to local project data. Our experience has been that 10 to 20 projects are adequate to achieve such tunings [9].

Local calibration is a simple tuning method that is supported by many tools[1]. Currently, our *variable reduction* methods requires more effort (i.e. some UNIX scripting) than local calibration. Figure 1 suggests that the extra effort may well be worthwhile, particularly when building models from a handful of projects. Also, we have found that it is easier to extrapolated costs from old projects to new projects with reduced variable sets [8, 10]. Neverthe-

[1] e.g. `http://sunset.usc.edu/available_tools/index.html`

less, variable reduction is *not* appropriate when there are business reasons to use all available variables (e.g. the three reasons described above).

3 Why Subtract Variables?

Having made the case for *keeping* variables, we now explore the other side of the coin. There are many reasons to *subtracting* variables. For example, from a *business perspective*, there are several situations where managing less variables is useful. For example:

- When monitoring an out-sourced project at a remote site, it is useful to have minimized the reporting requirements to just the variables that matter the most. Such a minimized reporting structure reduces the overhead in managing a contract.
- When assessing competing bids, a knowledge of what is most important can simplify the task of assessing the bids. For example:
  - All material relating to irrelevant variables can be safely ignored.
  - When doubling-checking the contractor's quotes against industrial standards, it is only necessary to check the variables that matter the most.

Also, from a *technical perspective*, there are good reasons to subtract variables:

Noise: Data collection was somehow flawed for a particular variable with the result that one or more variables are *noisy* (i.e. contains spurious signals not associated with variations to projects). Learning a cost estimation model is easier when the learner does not have to struggle with fitting the model to confusing noisy data.

Correlated variables: The reverse problem of noise is associated variables. If multiple variables are tightly correlated, then using all of them will diminish the likelihood that either variable attains significance. A repeated result in data mining is that subtracting some of the correlated variables increases the effectiveness of the learned model (the reasons for this are subtle and vary according to which particular learner is being used [4]).

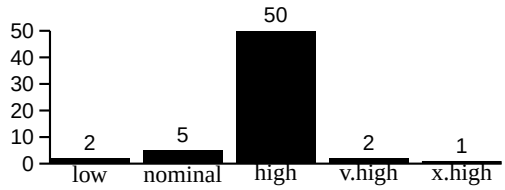

Figure 3: Distribution of software complexity *cplx* within the *nasa60* data set.

Under-sampling: The number of possible influences on a project is quite large and, usually, historical data sets on projects for a particular company are quite small. This motivates the subtraction of variables, even if those variables are theoretically useful.

To understand the under-sampling problem, consider two models for project *p02* (from Figure 1) that use two variables or four variables. Suppose that each variable takes one of five values (e.g. they are COCOMO variables with values *very high*, *high*, *nominal*, *low*, *very low*). The smaller model with two variables has an internal state space of $5^2 = 25$ and the larger model with four variables has an internal state space of $5^4 = 625$. Figure 1 records that project *p02* comprises 22 projects. These 22 projects could sample a large part of the two-variable model (up to $\frac{22}{25} = 88\%$). However, those same 22 projects can only ever sample a *very small fraction* ($\frac{22}{625} = 3.5\%$) of the four-variable model.

Figure 3 shows that under-sampling is not just a theoretical concern. In the *nasa60* data set, most projects cluster were rated as having a *high* complexity. Therefore, this data set would not support conclusions about the interaction of extra high complex projects with other variables. A learner would be wise to subtract this variable (and a cost modeling analyst would be wise to suggest to their NASA clients that they refine the local definition of "complexity").

Whatever the reason (business or technical), a repeated result in the data mining community is that simpler models with equivalent or higher performance can be built by *feature subset selection* (FSS) algorithms that intelligently subtraction of useless variables. For example, Kohavi and John's WRAPPER [7] builds models using an increasing number of variables; i.e. $N = 1, 2, 3, 4, ...$ For each N, there is a current *selected* set (initially, the empty set) and a set of remaining variables (those that have not

been selected). For each remaining variable, a new bigger set is tried containing the current set, plus one of the remaining variables. The set that yields the best learned model becomes the current set, N becomes $N + 1$ and the process continues. WRAPPER stops when either the remaining set is empty or there has been no significant improvement in the learned model for the last five additions (in which case, those last five additions are deleted).

WRAPPER is thorough and, according to Hall and Holmes, often yields the best results [4]. However, it can be too slow to run. It is simple to see why: there are a large number of possible subsets to explore. A naive search through all possible subsets of the 15 COCOMO-I variables would have to explore 32768 subsets. Happily, our study did not too long. The data sets of Figure 1 are quite small and our experiments only required around 20 minutes per data set.

The Hall and Holmes results were also negative about another widely used FSS technique: principle component analysis (PCA). FSS methods can be grouped according to:

- Whether or not they make special use of the target attribute in the data set such as "development cost";
- Whether or not they use the target learner as part of their FSS analysis.

PCA is unique since, unlike other FSS methods, it *does not* make special use of the target attribute. WRAPPER is also unique, but for different reasons: unlike other FSS methods, it *does* use the target learner as part of the FSS analysis. Hall and Holmes found that PCA was one of the worst performing FSS methods (perhaps because it ignored the target attribute) while WRAPPER was the best (since it can exploit its special knowledge of the target learner).

4 Case Study

To test the effectiveness of subtracting variables, we WRAPPER to select subtractable variables from the Figure 1 projects. Those projects were described in terms of the COCOMO set of variables.

COCOMO [1,2] is used for estimating software cost, effort and schedule. COCOMO helps software developers reason about the cost and schedule implications of their software decisions such as software investment decisions; setting project budgets and schedules; negotiating cost, schedule, and performance tradeoffs; making software risk management decisions, and making software improvement decisions.

COCOMO measures effort in calendar months where one month is 152 hours (and includes development and management hours). The core intuition behind COCOMO-based estimation is that as systems grow in size, the effort required to create them grows exponentially; i.e.

$$months = a * \left(KSLOC^b\right) * \left(\prod_i EM_i\right) \quad (1)$$

Here, EM_i is one of 15 *effort multipliers* such as *cplx* (complexity) or *pcap* (programmer capability). In the COCOMO-I model, a and b are domain-specific parameters and KSLOC is estimated directly or computed from a function point analysis. In COCOMO II, b was expanded to include *scale factors*:

$$b = 0.91 + \sum_j SF_j$$

where SF_j is one of five *scale factors* that exponentially influence effort. Examples of scale factors include *pmat* (process maturity) or *resl* (attempts to resolve project risks).

Various learning methods have been applied to COCOMO. Elsewhere [10], we have seen that best results come from transforming the COCOMO-I equation of Equation 1 into the *linearized model* of Equation 2:

$$LN(effort) = b*LN(Size)+LN(EM_1)+LN(EM_2)+\ldots \quad (2)$$

For COCOMO-II, the linearized model is:

$$\begin{aligned} LN(effort) &= SF_1 * LN(Size) + SF_2 * LN(Size) + ... \\ & \quad LN(EM_1) + LN(EM_2) + \ldots \end{aligned} \quad (3)$$

At each step in the WRAPPER, some learner built a model from the current set of variables. Elsewhere we have tried various methods for such learning. In those experiments, simple linear regression on over the linearized COCOMO models did as well as any other method [10].

4.1 Data

This study used two data sets described using COCOMO I variables and two using COCOMO-II variables:

- *Coc81* comes from the COCOMO-I text [1] and includes data from a variety of domains including engineering, science, financial, etc.
- The *cii00* data set is the COCOMO-II data.
- The *cii04* data set includes the 72 projects from *cii00* developed after 1990, plus 47 new projects.
- *Nasa60* comes from 20 years of NASA projects and is recorded using the COCOMO-I variables. This data comes from multiple *projects* developed at NASA centers at different geographical *locations*; performing different *tasks* such as ground data receiving, flight guidance,etc.

Figure 1 shows results from the above four data sets as well as several subsets of *nasa60*. Those subsets come from three NASA centers *c01,c02,c03*; three NASA projects *p02,p03,p04*; and two NASA tasks *t02,t03*. For reasons of confidentiality, the exact details of those centers, tasks, and projects cannot be disclosed. The other centers, projects, and tasks from *nasa60* were not included for a variety of pragmatic reasons (e.g. suspicious repeated entries suggesting data entry errors, too few examples for generalization, etc).

Of these data sets, *coc81* describes projects from before 1982; *ci04* contain data from the most recent projects; and the NASA data sets (*nasa60, pX,cX,tX*) described projects newer than *coc81* and before *ci04*.

4.2 Using the Data

For each dataset, our experiments were in two parts: *feature set generation*, followed by *feature set subtraction*.

Randomization was used extensively in our experiments. Many algorithms have an *order effect* such that their performance changes dramatically if the inputs are re-ordered. Kermer reported order effects in his analysis data from 15 projects: if training was restricted to 9 particular projects, the learning was far more successful [5]. Randomization avoids such order effects.

In *feature set ordering*, the WRAPPER was called on 10 randomly sub-samples of the data set. Each sub-sample contained 90% of the data (selected at random).

coc81			p02		
FSx	variable	# times selected	FSx	variable	# times selected
10	loc	10	7	loc	10
9	sced	10	6	turn	7
9	pcap	10	6	lexp	7
9	time	10	5	time	4
8	virt	9	4	modp	3
7	cplx	6	3	data	2
6	modp	5	3	tool	2
6	acap	5	3	sced	2
6	rely	5	2	rely	1
5	vexp	4	2	vexp	1
4	tool	3	2	cplx	1
4	data	3	1	aexp	0
3	aexp	2	1	pcap	0
3	stor	2	1	virt	0
2	lexp	1	1	acap	0
1	turn	0	1	stor	0

Figure 4: Some of the generated features sets.

Variable were then grouped together according to how often they were *selected* in the 10 sub-samples. All the variables that were selected with the same frequency were placed together into the same *feature set* FSx. The feature sets are then ordered by their frequency counts and then numbered. To these feature sets, an extra sets are added: "All" (holding all the variables). For example the table of Figure 4 shows the feature sets found in *coc81* and *p02*. Note that different data sets generate different feature sets (we will comment on this, below). Note also that, in all our experiments, loc (lines of code) was always selected every time.

After *generation*, came *feature set subtraction*. 30 times, we randomized the order of the data. Next, starting with all the features, we subtracted each feature set in turn. For each set, we conducted a hold-out experiments to assess the value of that particular subset of the variables. In those hold-outs, we divided the project data described using some FSx set into a $\frac{2}{3} : \frac{1}{3}$ subsets, trained on the $\frac{2}{3}rd$ subset and tested on the $\frac{1}{3}rd$ test subset. The whole point of cost models is that they can be used to estimate the cost of *new* projects. Such hold-out experiments ensure that we are assessing the learned model on the *new data* in the test subset. For each set, the mean and standard deviation of the performance score on the test set was collected. Scores from different sets were then compared using t-tests:

- The performance of each model was scored using

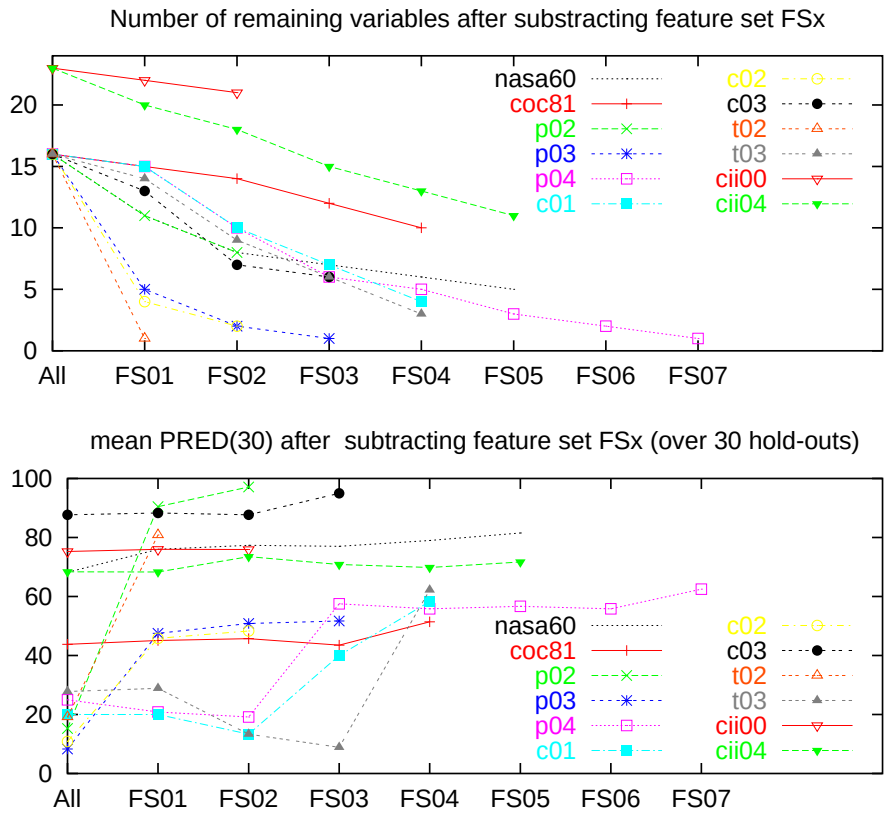

Figure 5: Feature sets. The results from each data set stop at the point of greatest "total wins - total losses".

PRED(30); i.e. the percentage of estimates in the test set that are within 30% of the actual values. We have found PRED(30) easier to explain to business than alternate measures. Also, landmark high-water marks in software cost estimation report their results in terms of PRED(30) [3].

- To conduct the t-tests, the mean and standard deviation of the different PRED(30)s seen in all pairs of FSx and FSy were computed. FSx and FSy were said to "tie" if there was no statistical difference (at the $\alpha = 0.05$ level) detectable. If the comparison did not "tie", then the means were numerically compared to compute "win"s and "loss"es. The "best" FSx set was selected as the one with the highest "total wins-total losses" score.

All the plots in Figure 5 start at "All" and stop at the feature subset with the maximum 'total wins - total loses" (and if two sets score the same "total wins - total loses", then the one with highest mean was selected as the winner). Figure 1 was generated by comparing the "All" mean PRED(30) to the "best" mean PRED(30).

As shown in the top "number of remaining variables" plot of Figure 5, in the usual case, most of the variables are subtracted. Usually, the data sets were pruned back to less than five variables. Three data sets (*p03, p04, t02*) got pruned back to just one variable (lines of code). The larger data sets (*cii04,cii00,coc81,nasa60*) were pruned back the least. This is hardly surprising: the more data the more interesting the inter-relationships and the less the win in subtracting variables.

data set	#projects	standard deviation on PRED(30) in 30 hold-outs before	after	$\frac{after}{before}$
nasa60	60	11%	8%	68%
cii00	161	9%	7%	83%
coc81	63	11%	10%	89%
cii04	119	5%	5%	94%
	mean	9%	7%	83%

data set	#projects	before	after	$\frac{after}{before}$
p02	22	12%	6%	49%
c03	31	12%	6%	51%
t02	14	26%	19%	74%
c01	13	21%	21%	100%
p03	12	15%	16%	105%
p04	14	21%	26%	126%
c02	12	17%	24%	139%
t03	10	22%	32%	150%
	mean	18%	19%	99%

Figure 6: Effects on standard deviation of variable subtraction.

The bottom "mean PRED(30)" plot of Figure 5 shows results from 30 *hold-out* experiments conducted for each features set. One interesting feature of this "mean" plost is that "best" PRED never results from using "All" variables. This result endorses the merits of variable subtraction: in terms of generating models with high PRED, we have no evidence here that there is any down side to always conducting feature subtraction (but recall our previous remarks- sometimes there are business reasons for *not* subtracting features).

When we looked at the standard deviations in our 30 hold-outs (see Figure 6), we found no extra evidence for or against variable subtraction. For larger data sets, variable substraction always reduced standard deviation. For smaller data sets, there was no clear pattern: subtraction changed standard deviation anywhere from 49% to 150% and, on average, there was little change (last=99% of first).

5 Related Work

Variable subtraction (a.k.a. feature subset selection) has been widely studied in the data mining literature (e.g. [4, 7]). Mostly those studies have focused on discrete classes. To the best of our knowledge, the only other work on variable reduction and for continuous class cost estimation data is Kirsopp & Shepperd's (K&S) case base reasoning (CBR) work [6].

In CBR, decisions about the current case are then made by studying similar historical cases. A common technique is to use some form of distance metric to extract the k-th nearest neighbors from the current case within a case library. Some extrapolation mechanism is then applied to analyze the nearest neighbors and make some decision about expected properties of the current case.

K&S explores CBR for cost estimation using feature subset selection using two data sets:

- The *small* data set containing 10 variables and data from 77 projects;
- The *large* data set containing 48 variables and data from 407 projects.

Like us, K&S found that FSS significantly improves effort estimation. However, and contrary to Figure 1, they found that FSS improved effort estimation more in the *large* data sets than in the *small* data set. There are several possible explanations of why our results are so different to those of K&S. Firstly, the K&S definition of "small" is much larger than our definition. For example, our "small" data sets contain 10 to 22 projects while K&S's *small* data set contains 77 projects. It is possible that if K&S ran their toolkit over our very small data sets, they might find a different effect. Secondly, the studied algorithms are different: K&S use CBR while we use regression over a parametric COCOMO models. It is possible that CBR is confused by smaller data sets. Thirdly, we are using different data sets and to really compare K&S with our results, we need to run COCOMO data through their toolkit.

6 Discussion

The benefits of variable subtraction should be carefully weighed against the business implications of subtracting seemingly irrelevant variables:

- Cost models should contain all available variables when business knowledge argues for retention.
- Cost models should reject some variables (selected by WRAPPER) when the goal is to improve effort

estimation. As seen in Figure 1, for small data sets (30 projects or less), the improved effort estimation can be quite dramatic. This increased improvement in the smaller data sets is an important result since, typically, organizations only have data on a small number of projects.

When discussing these results, we are often asked why we have not run these experiments on more data sets. Accessing such further data sets is a non-trivial task. It is hard enough getting *any* data from *any* organization, let alone data in the COCOMO format. There is a good reason for this. Software projects are notoriously difficult to control. Recall the 2001 report of the Standish group that described a software industry where 23% of projects totally fail, 28% meet all expectations, and the remaining 49% were significantly challenged in some way (e.g. over-budget, over the time estimate, fewer features and functions that initially specified) [12]. Corporations are therefore reluctant to expose their own less-successful software development practices to public scrutiny.

Nevertheless, our goal is to apply our methods to more data from more projects. For example, we plan to run our data sets through the K&S toolkit. Also we are teaming with the University of Ottawa to extend the PROMISE repository of public-domain data sets relating to software engineering [11]. For example, if the reader want us to try our techniques on their data, they just need to submit it to that repository[2] (perhaps with some anonymization of any business-critical information).

We end with an important methodological note. Figure 4 showed that different data sets reject different variables. Hence it would be a mistake to interpret our results as "variable X is not relevant in all domains". Rather, the best variables in different domains should be found by separate runs of the WRAPPER.

References

[1] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.

[2] Barry Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, A. Winsor Brown, Sunita Chulani, and Chris Abts. *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.

[3] 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.

[4] M.A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering*, 15(6):1437– 1447, 2003.

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

[6] C. Kirsopp and M. Shepperd. Case and feature subset selection in case-based software project effort prediction. In *Proc. of 22nd SGAI International Conference on Knowledge-Based Systems and Applied Artificial Intelligence, Cambridge, UK*, 2002.

[7] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[8] T. Menzies, D. Port, Z. Chen, J. Hihn, and S. Stukes. Automatically learning softwate cost extrapolations. 2005. Submitted, IEEE ASE, 2005, Available from `http://menzies.us/pdf/05learncost.pdf`.

[9] 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`.

[10] Tim Menzies, Zhihao Chen, Dan Port, and Jairus Hihn. Simple software cost estimation: Safe or unsafe? In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies.us/pdf/05safewhen.pdf`.

[11] J. Sayyad Shirabad and T.J. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology

[2] `http://promise.site.uottawa.ca/SERepository/dataset-software-donation-page.html`

and Engineering, University of Ottawa, Canada, 2005. Available from http://promise.site.uottawa.ca/SERepository.

[12] The Standish Group Report: Chaos 2001, 2001. Available from http://standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf.