Finding the Right Data for Software Cost Modeling

Zhihao Chen, *University of Southern California*
Tim Menzies, *Portland State University*
Daniel Port, *University of Hawaii*
Barry Boehm, *University of Southern California*

Strange to say, when building a software cost model, sometimes it's useful to ignore much of the available cost data.

Good software cost models can significantly help software project managers. With good models, project stakeholders can make informed decisions about how to manage resources, how to control and plan the project, or how to deliver the project on time, on schedule, and on budget.

Off-the-shelf "untuned" models have been up to 600 percent inaccurate in their estimates.[1] So, the wise manager uses a cost model built from local data. But what data should you use to build a good cost model? Real-world data sets, such as those coming from software engineering projects, often contain noisy, irrelevant, or redundant variables.

Before getting the data, it's hard to know what parts of the data are most important. However, once a database is available, you can use automatic tools to prune the data to the most important values. Therefore, we propose a change to current practice. Often, cost models are built using all available data. We propose that cost modelers should perform data-pruning experiments after data collection and before model building. Such pruning experiments are simple and fast.

The basic process

We start with a table of historical data divided into columns and rows. Each column is a different variable describing some aspect of a software project. Each row shows data from different software subsystems, so one project can contribute many rows. For example, in table 1, the Variables columns show 17 variables that can be collected from a software project, and the rows refer to three subsystems from two projects.

We can prune the data in such a table by removing rows or columns. *Row pruning* (also called *stratification*) collects rows from related projects; different cost models are learned from these different subsets.

Column pruning (also called *feature subset selection*) first sorts the columns left to right according to their usefulness—that is, how well a column's variable predicts for the target variable (in our case, software development effort). The pruning then proceeds left to right across the sorted columns, each time removing

0740-7459/05/$20.00 © 2005 IEEE

some less-useful left-hand columns. At each pruning step, a cost model is learned from the remaining columns.

Row pruning has demonstrable benefits. For example, Martin Shepperd and Chris Schofield report row-pruning experiments that improved estimator performance up to 28 percent.[2] However—and this is our point here—we find that much larger improvements result from pruning both rows and columns. For example, our experiments include a data set where estimator performance went from 15 percent to 97 percent.

Furthermore, these large improvements occurred when we pruned most of the columns away. For example, column pruning removed 65 percent of all columns (on average). Surprisingly, when building a software cost model, it's usually useful to ignore over one-half of the available cost data.

More important, row-and-column pruning leads to the largest improvements in estimator performance in the smallest training sets (fewer than 30 examples). This result has tremendous practical significance. Modern software practices change so rapidly that most organizations can't access large databases of relevant project data. Our results suggest that this isn't necessarily a problem, provided models are learned via row and column pruning.

For a look at a different application of data pruning to software cost estimation, see the sidebar.

Pruning: Why?

Let's look in more detail at why you would want to prune data. The case for pruning rows is simple. Software projects aren't all the same. For example, real-time safety-critical systems are very different from batch financial processors. Given a database of different kinds of software, it's just good sense to divide the rows into different project types and learn different cost models for each type. Then, in the future, managers can use different cost models depending on what type of software they're developing.

The case for pruning columns is slightly more complex. If a learned cost model uses all the variables in the database, then the only way to use that cost model on future subsystems is to collect information on all those variables. In many business situations, the cost of reaching some goal is a function of how much data you must collect or monitor along the way. If the learned model uses only some of the variables, then using that model in the future means collecting less data. This would be useful in several scenarios. For example, if you're monitoring an outsourced project at a remote site, it's useful to minimize the reporting requirements to the most important variables. Such a reporting structure reduces the overhead in managing a contract.

Good technical reasons for subtracting variables also exist:

Undersampling

The number of possible influences on a project is quite large, and, usually, historical

Table 1

Some data for a software engineering project

		Variables					
Project	Subsystem	1 Analyst ability	2 Process maturity	3 Required reliability	... ...	17 Lines of code	Development effort (mos.)
$proj_1$	DBapi	High	Low	High	...	100,000	467
$proj_1$	GUI	High	Low	Low	...	200,000	847
$proj_2$	Guidance	High	High	Nominal	...	50,000	174

Related Work on Data Pruning and Software Cost Modeling

Data-pruning methods evolved within the data-mining community. While researchers have extensively explored them,[1,2] little prior work exists on column pruning for software cost modeling.

To the best of our knowledge, the only research similar to ours was a limited experiment by Colin Kirsopp and Martin Shepperd.[3] As with our study, they found that column pruning significantly improves effort estimation. However, their experimental base was much more restrictive than ours (they ran only two data sets; we ran 15). Also, unlike our work, their experiment isn't reproducible. Their data sets aren't public domain, while you can download all our COCOMO-I data sets from the PROMISE repository.[4]

References

1. M.A. Hall and G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 6, 2003, pp. 1437–1447.
2. R. Kohavi and G.H. John, "Wrappers for Feature Subset Selection," *Artificial Intelligence*, vol. 97, nos. 1–2, 1997, pp. 273–324.
3. C. Kirsopp and M. Shepperd, "Case and Feature Subset Selection in Case-Based Software Project Effort Prediction," *Proc. 22nd SGAI Int'l Conf. Knowledge-Based Systems and Applied Artificial Intelligence*, 2002.
4. J. Sayyad Shirabad and T.J. Menzies, *The PROMISE Repository of Software Engineering Databases*, School of Information Technology and Eng., Univ. of Ottawa, 2005; http://promise.site.uottawa.ca/SERepository.

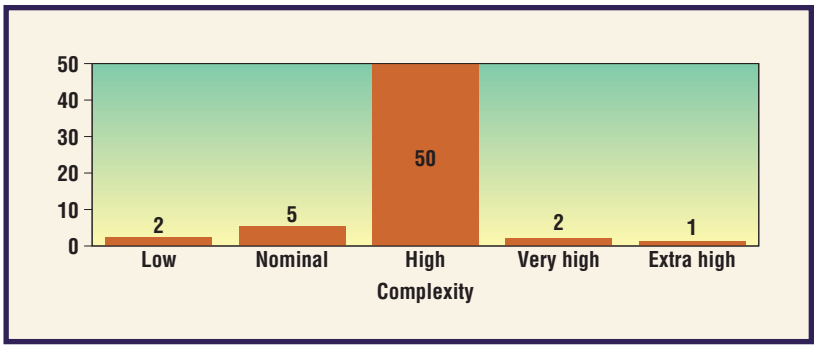


Figure 1. Distribution of software complexity in *na*60, a NASA project data set.

data sets on projects for a particular company are quite small. So, a variable that's theoretically useful might be practically useless. For example, figure 1 shows how nearly all the NASA projects in the *na*60 data set were rated as having high complexity. Therefore, this data set wouldn't support conclusions about the interaction of, say, projects having very high complexity with other variables. A learner would be wise to subtract this variable (and a cost-modeling analyst would be wise to suggest to his or her NASA clients that they refine the local definition of complexity).

Reducing variance

Alan Miller has extensively surveyed column pruning for linear models and regression.[3] His survey includes a strong argument for column pruning: the variance of a linear model learned by minimizing least-squares error decreases as the number of columns in the model decreases. That is, the fewer columns there are, the more restrained the model predictions are.

Irrelevancy

Sometimes, modelers have incorrect beliefs about what variables affect some outcome. In this case, they might add irrelevant variables to a database. Without column pruning, a cost model learned from that database might contain these irrelevant variables. Anyone trying to use that model in the future would then be forced into excessive data collection.

Noise

Learning a cost estimation model is easier when the learner doesn't have to struggle with fitting the model to confusing noisy data (that is, data containing spurious signals not associated with variations to projects). Noise can come from many sources, such as clerical errors or missing data. For example, organizations that build only word processors might have little data on software projects with high-reliability requirements.

Correlated variables

If multiple variables are tightly correlated, using all of them will diminish the likelihood that either variable attains significance. A repeated result in data mining is that pruning away some correlated variables increases the learned model's effectiveness (the reasons for this are subtle and vary according to which learner you use[4]).

Pruning: How?

To prune columns, we use the Wrapper variable-subtraction algorithm, which selects combinations of columns and asks some learner to build a cost model using just those columns.[5] The Wrapper then grows the selected columns and checks if a better model comes from learning over the larger set of columns.

The Wrapper stops when there are no more columns to select or when the learned model has seen no significant improvement over the last five additions (in which case, the Wrapper discards those additions). Technically speaking, this is a forward-select search with a "stale" parameter set to 5.

The Wrapper is thorough, but it's theoretically quite slow because (in the worst case) it must explore all subsets of the available columns. However, all the data sets in our study were quite small. Our experiments required only approximately 20 minutes per data set.

We use the Wrapper because other experiments by other researchers strongly suggest that it's superior to many column-pruning methods. For example, Mark Hall and Geoffrey Holmes compare this algorithm to several methods including *principal component analysis*, a widely used technique.[4] You can group these methods according to

- whether they make special use of the target attribute in the data set such as "development cost" and
- whether they use the target learner as part of their pruning.

PCA is unique because it doesn't make special use of the target attribute. The Wrapper is

Table 2

Cocomo-I effort multipliers and their numeric values*

Effort multiplier	Definition	Complexity: Very low	Low	Nominal	High	Very high	Extra high
ACAP	Analysts' capability	1.46	1.19	1.00	0.86	0.71	—
PCAP	Programmers' capability	1.42	1.17	1.00	0.86	0.70	—
AEXP	Application experience	1.29	1.13	1.00	0.91	0.82	—
MODP	Modern programming practices	1.24	1.10	1.00	0.91	0.82	—
TOOL	Use of software tools	1.24	1.10	1.00	0.91	0.83	—
VEXP	Virtual-machine experience	1.21	1.10	1.00	0.90	—	—
LEXP	Language experience	1.14	1.07	1.00	0.95	—	—
SCED	Schedule constraint	1.23	1.08	1.00	1.04	1.10	—
DATA	Database size	—	0.94	1.00	1.08	1.16	—
TURN	Turnaround time	—	0.87	1.00	1.07	1.15	—
VIRT	Machine volatility	—	0.87	1.00	1.15	1.30	—
STOR	Main-memory constraint	—	—	1.00	1.06	1.21	1.56
TIME	Time constraint for CPU	—	—	1.00	1.11	1.30	1.66
RELY	Required software reliability	0.75	0.88	1.00	1.15	1.40	—
CPLX	Process complexity	0.70	0.85	1.00	1.15	1.30	1.65

* *Increasing the top seven multipliers or decreasing the bottom seven multipliers will decrease effort.*

also unique, but for different reasons: unlike other pruning methods, it uses the target learner as part of its analysis. Hall and Holmes found that PCA was one of the worst-performing methods (perhaps because it ignored the target attribute) while the Wrapper was the best (because it can exploit its special knowledge of the target learner).

Cost modeling with Cocomo

Before pruning data, we first need to understand how cost models might use that data. This study uses Cocomo (*Co*nstructive *Co*st *Mo*del) for cost modeling. 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 trade-offs;
- making software risk management decisions; and
- making software improvement decisions.

One advantage of Cocomo (which is why we use it) is that unlike many other costing models such as SLIM (Software Lifecycle Management) or SEER-SEM (System Evaluation and Estimation of Resources, Software Estimation Model), it's an open model with numerous published data.[6,7]

Two versions of Cocomo exist. In going from the 1981 Cocomo-I model[6] to the 2000 Cocomo-II model,[7] one parameter, turnaround time, was dropped to reflect the almost-universal use of interactive software development. Cocomo-II also dropped the modern-programming-practices parameter in favor of a more general process-maturity parameter. But Cocomo-II added several parameters to reflect such factors as development for reuse, multisite development, architecture and risk resolution, and team cohesion. The Cocomo-II book also provides capabilities and guidelines for an organization to add new parameters, reflecting its particular situations.[7]

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. More specifically, equation 1 shows the Cocomo-I model:[6]

$$months = a * \left(KSLOC^{b}\right) * \left(\prod_{j} EM_{j}\right) \qquad (1)$$

Here, EM_j is one of the 15 Cocomo-I *effort multipliers* such as CPLX (complexity) or

Pruning is most important for small data sets.

PCAP (programmer capability). Table 2 defines these effort multipliers and lists their numeric values. COCOMO-II includes 17 effort multipliers.

In the COCOMO-I model, *a* and *b* are domain-specific variables and KSLOC (thousands of lines of noncommented source code) is estimated directly or computed from a function point analysis. COCOMO-II 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).

A standard method for assessing COCOMO performance is Pred(30). You calculate Pred(30) from the *relative error* (RE), which is the relative size of the difference between the actual and estimated value:

$$RE = (estimate - actual)/actual$$

The *mean magnitude of the relative error* (MMRE) is the average percentage of the absolute values of the relative errors over an entire data set. Pred(*N*) reports the average percentage of estimates that were within *N* percent of the actual values. If a data set has rows, then

$$\text{Pred}(N) = \frac{100}{R}\sum_i^R \begin{cases} 1 & \text{if } RE_i \le \frac{N}{100} \\ 0 & \text{otherwise} \end{cases}$$

For example, Pred(30) = 50 percent means that one-half of the estimates are within 30 percent of the actual. We report results in terms of Pred(*N*), not MMRE. This is a pragmatic decision—we've found Pred(*N*) easier to explain to business users. Also, there are more Pred(*N*) reports in the literature. This is perhaps due to the influence of the COCOMO researchers who reported their 1999 landmark study using Pred(*N*).[8] Furthermore, we report Pred(30) results because the major experiments of that 1999 study also used Pred(30).

To use linear least-squares regression, the most widely used and simplest modeling method, it's common to transform the COCOMO model into a linear model by taking the logs of equation 1:

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

If we use equation 2, then before computing Pred(*N*), we must convert the estimated effort back from a logarithm.

Case study data

Our study used data sets in both the COCOMO-I and COCOMO-II formats. We used these COCOMO-I data sets:

- *coci* includes data from a variety of domains including engineering, science, and finance.
- *na*60 comes from 20 years of NASA projects.

The *na*60 data includes these subsets:

- *c*01, *c*02, and *c*03 store data from three NASA geographical locations.
- *p*02, *p*03, and *p*04 store data from three NASA projects.
- *t*02 and *t*03 store data from two tasks such as ground-data receiving and flight guidance.

For reasons of confidentiality, we can't disclose the exact details of these locations, tasks, and projects. We didn't include the other locations, projects, and tasks from *na*60 for pragmatic reasons (for example, suspicious repeated entries suggesting data entry errors or too few examples for generalization).

We used two COCOMO-II data sets:

- The *cii*0 data set was used to build the COCOMO-II model.
- The *cii*4 data set includes the 72 projects from *cii*0 developed after 1990, plus 47 new projects.

The COCOMO-II data isn't published because it was collected on condition of confidentiality with the companies supplying the data. Further research must be conducted in terms of the same conditions.

Of these data sets, *coci* describes projects from before 1982, *cii*4 contains data from the most recent projects, and the NASA data sets (*na*60, *pX*, *cX*, and *tX*) describes projects newer than *coci* and before *cii*4. Also, the COCOMO-I data sets have the 15 effort multipliers

in table 2; the COCOMO-II data sets have 24 effort multipliers.

For experimental purposes, we used these groups:

- *call* combines all the center data—that is, *call* = *c*01 + *c*02 + *c*03.
- *tall* combines all the task data—that is, *tall* = *t*02 + *t*03.
- *pall* combines all the project data—that is, *pall* = *p*02 + *p*03.

Experimental method

To find each column's relative value, we ran the Wrapper 10 times (each time using a randomly selected 90 percent of the rows). We then set a column's value to the number of times the Wrapper selected that column.

Once the Wrapper ordered the columns, we randomized the order of the rows and started column pruning. To ensure statistical validity, we repeated randomization (followed by column pruning) 30 times.

For each repeat, at each pruning stage, we removed the lowest-value column. We then divided the rows in the remaining columns into training and test sets (each time using a randomly selected 67 percent of the rows for the training set). A cost model was learned (using linear regression) from the training set and then assessed, using Pred, on the test set.

Once the 30 repeats completed, we selected the best model by looking at the mean and standard deviation of model performance at each pruning step. The best model was the one that t-tests confirmed outperformed all other states of the column pruning. Our scripts then automatically computed that model's mean value over the 30 repeats.

We fully automated this process using a tool consisting of Unix scripts that control the WEKA (Waikato Environment for Knowledge Analysis) Java data-mining library.[9] WEKA includes a linear-regression learner and an implementation of the Wrapper. The tool runs on a standard Linux installation, and the whole system is available from the authors.

Results

Figure 2 shows the results of our column pruning. The red and green lines show the number of columns in our data sets before and after pruning. The Before values are 22 for the COCOMO-II data sets (*cii*0 and *cii*4) and 15 for the COCOMO-I data sets. The blue line shows the percentage of the Before columns that pruning removed. For example, pruning removed few columns from *cii*0 but most columns from *p*03. As we mentioned earlier, we pruned on average over 65 percent of the columns. Sometimes, the pruning was quite heavy—over 80 percent of the columns.

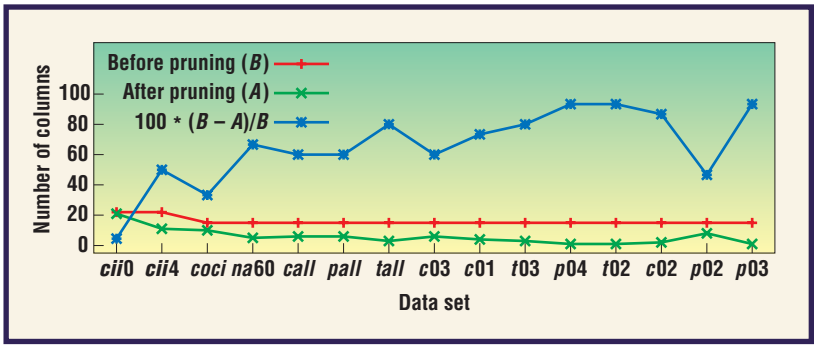


Figure 2. The number of columns (variables describing software projects) that pruning removed from each data set.

A concern with such large-scale pruning is that the resulting models might be somehow substandard. This proved not to be the case. Figure 3 shows the Pred(30) results for the pruned data sets. This figure shows mean values for 30 experiments where we tested the learned model on rows not seen during training. The red lines show the mean Pred(30) for using all the columns. These are the baseline results for learning cost models before applying any pruning method. The green lines show the best mean Pred(30) after automatic column pruning. The difference between the red and the green lines is the improvement that pruning produced.

Figure 3 sorts the data sets by pruning method into three plots. Each plot sorts the data sets from left to right in increasing value of (*after* – *before*)/*before*.

The left plot, "Pruning just columns," shows the largest data sets, which combine project information from many sources (that is, *cii*0, *cii*4, *coci*, and *na*60). These data sets don't contain data from similar sources. So, we performed no row pruning on these data sets.

The right plot, "Pruning columns and many rows," shows the eight data sets that have been heavily stratified into specific NASA centers, projects, or software tasks.

The middle plot, "Pruning columns and some rows," shows somewhat-stratified data sets that combine data from either all the NASA centers, all the NASA projects, or all

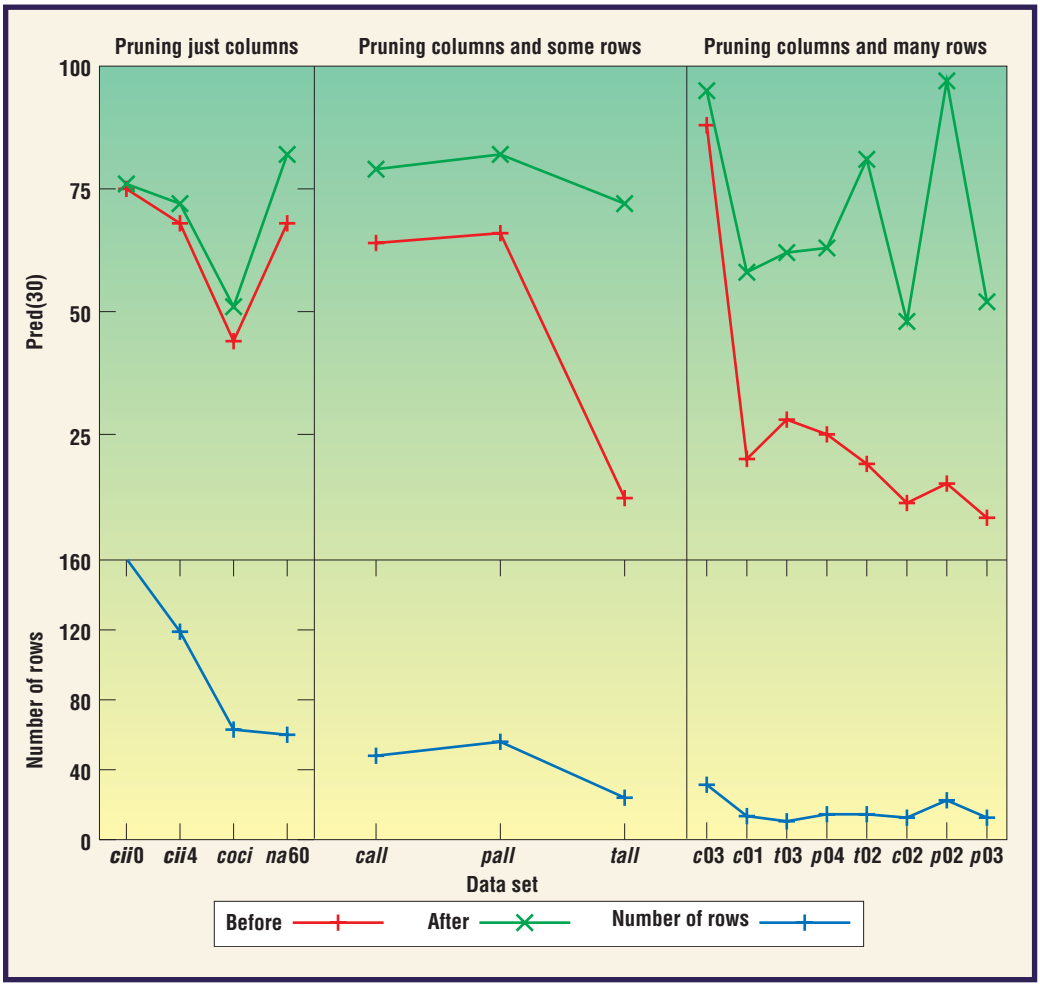


Figure 3. How different pruning methods affected the Pred(30) measure of predictive accuracy.

the NASA tasks. This group samples a point halfway between the unstratified data sets on the left and the heavily stratified data sets on the right.

Figure 3 reveals these interesting results:

- Pruning always improved estimation effectiveness. That is, in all our case studies, it was always useful to ignore a portion of the available data.
- Column pruning by itself can improve Pred somewhat. However, column pruning combined with row pruning can dramatically improve effort estimation.
- With one exception (*c*03), the general trend across the three graphs is clear: as data set size shrinks, the improvement increases. That is, pruning is most important for small data sets.

When not to prune

Although column pruning is clearly useful, sometimes you can't or shouldn't apply it. First, our variable-subtraction methods require a historical database of projects. If no such database exists, our column-pruning techniques won't work.

Second, even if a historical database exists and our techniques suggest pruning variable *X*, then it might still be important to ignore that advice. If a cost model ignores certain effects that business users believe are important, those users might not trust that model. In that case, even if a variable has no noticeable impact on predictions, you should leave it in the model. By doing this, you're acknowledging that, in many domains, expert business users hold in their head more knowledge than might be available in historical databases.

A third reason not to prune variables is that you still might need them. For example, our experiments often subtracted over one-half of the attributes in a COCOMO-I model while (usually) improving effort estimation. However,

suppose that a user needs some of the pruned variables to make a business decision. Because the reduced model has no information on those variables, the user would have to resort to other information to make the decision.

So, you should use column pruning with some care. If no historical data exists for learning specialized data sets, then managers should use the general background knowledge in COCOMO. The 1981 regression coefficients of COCOMO-I or the updated coefficients of COCOMO-II[7] 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 factor, the increase in cost between a CMM Level 3 project and a CMM Level 4 project, each containing N lines of code, is $N^{3.13}/N^{1.56}$. With this estimate in hand, a business user could then make an assessment about the cost of increased software process maturity versus that increase's benefits.

If historical data from the local site is available, managers could tune the general COCOMO background knowledge by adjusting the coefficients in the COCOMO equations. COCOMO-I and COCOMO-II contain 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.[10]

Local calibration is a simple tuning method that many tools support (for examples, visit http://sunset.usc.edu/available_tools/index.html). Currently, our toolkit's methods require more effort (that is, some Unix scripting) than local calibration. Figure 3 suggests that the extra effort might well be worthwhile, particularly when you're building models from a handful of projects. Also, we have other reasons for preferring reduced variable sets to local calibration. Elsewhere,[11] we've found it easier to extrapolate costs from old projects to new projects using the reduced sets. Nevertheless, as we mentioned earlier, column pruning isn't appropriate when there are business reasons to use all available variables.

Our main goal here is to encourage more column pruning in cost modeling, particularly when dealing with very small data sets. Our more general goal is to encourage repeatable, refutable, and improvable experiments in software engineering. To that end, as much as possible, we use public-domain tools and public-domain data sets. So for these experiments we used an open-source cost model (COCOMO) and, as much as possible, publicly available data. You can download all the COCOMO-I data sets we used from the PROMISE repository.[12] We urge other researchers to produce more results based on open source models and data sets.

Acknowledgments

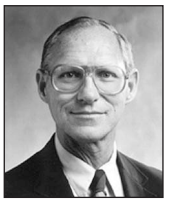
Helen Burgess offered invaluable editorial assistance. The anonymous reviewers' advice helped clarify an earlier draft of this article, which appeared previously in a workshop publication.[13] This article extends that earlier draft in two ways. First, it explores more data (here we analyze double the number of COCOMO-I projects and add two new COCOMO-II data sets). Second, it includes an expanded discussion on column pruning's business implications.

About the Authors

Zhihao Chen is a research assistant at the University of Southern California's Center for Software Engineering and a PhD candidate in the university's Computer Science Department. His research interests are system software engineering, model development, and integration in general. He received his masters in computer science from the South China University of Technology. Contact him at zhihaoch@usc.edu.

Tim Menzies is an associate research professor at Portland State University and works with NASA on software quality. His recent research concerns modeling and learning, with a particular focus on lightweight modeling methods. He received his PhD in knowledge acquisition and artificial intelligence from the University of New South Wales. Contact him at tim@timmenzies.net.

Daniel Port is an assistant professor of IT management at the University of Hawaii at Manoa and a cofounder of the university's proposed Center for Strategic Software Engineering. He's also a visiting scholar at the University of Southern California's Center for Software Engineering. His primary research activities lie in strategic and economic-based software engineering. He received his PhD in applied mathematics from the Massachusetts Institute of Technology. Contact him at dport@hawaii.edu.



Barry Boehm is the director of the University of Southern California's Center for Software Engineering. His research interests include software process modeling, software requirements engineering, software architectures, software metrics and cost models, software engineering environments, and knowledge-based software engineering. He received his PhD in mathematics from UCLA. He's a fellow of the American Institute of Aeronautics and Astronautics, the ACM, and the IEEE and is a member of the National Academy of Engineering. Contact him at boehm@sunset.usc.edu.

ADVERTISER / PRODUCT INDEX

NOVEMBER/DECEMBER 2005

Advertiser / Product	Page Number
Addison-Wesley	103
Auerbach Publications	104
Carnegie Mellon Software Engineering Institute	**11**
ICRE 2006	**16**
John Wiley & Sons, Inc.	105
MIT Press	**7**
Morgan Kaufmann Publishers	104
SD West 2006	**Cover 4**

Boldface denotes advertisements in this issue.

Advertising Personnel

Marion Delaney
IEEE Media, Advertising Director
Phone: +1 212 419 7766
Fax: +1 212 419 7589
Email: md.ieeemedia@ieee.org

Marian Anderson
Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown
IEEE Computer Society,
Business Development Manager
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Advertising Sales Representatives

Mid Atlantic (product/recruitment)
Dawn Becker
Phone: +1 732 772 0160
Fax: +1 732 772 0161
Email: db.ieeemedia@ieee.org

New England (product)
Jody Estabrook
Phone: +1 978 244 0192
Fax: +1 978 244 0103
Email: je.ieeemedia@ieee.org

New England (recruitment)
Robert Zwick
Phone: +1 212 419 7765
Fax: +1 212 419 7570
Email: r.zwick@ieee.org

Southeast (product)
Bob Doran
Phone: +1 770 587 9421
Fax: +1 770 587 9501
Email: bd.ieeemedia@ieee.org

Southern CA (product)
Marshall Rubin
Phone: +1 818 888 2407
Fax: +1 818 888 4907
Email: mr.ieeemedia@ieee.org

Southeast (recruitment)
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynntom@mindspring.com

Midwest/Southwest (recruitment)
Darcy Giovingo
Phone: +1 847 498-4520
Fax: +1 847 498-5911
Email: dg.ieeemedia@ieee.org

Midwest (product)
Dave Jones
Phone: +1 708 442 5633
Fax: +1 708 442 7620
Email: dj.ieeemedia@ieee.org

Will Hamilton
Phone: +1 269 381 2156
Fax: +1 269 381 2556
Email: wh.ieeemedia@ieee.org

Joe DiNardo
Phone: +1 440 248 2456
Fax: +1 440 248 2594
Email: jd.ieeemedia@ieee.org

Southwest (product)
Josh Mayer
Phone: +1 972 423 5507
Fax: +1 972 423 6858
Email: josh.mayer@wageneckassociates.com

Northwest (product)
Peter D. Scott
Phone: +1 415 421 7950
Fax: +1 415 398 4156
Email: peterd@pscottassoc.com

Northwest/Southern CA (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Connecticut (product)
Stan Greenfield
Phone: +1 203 938 2418
Fax: +1 203 938 3211
Email: greenco@optonline.net

Japan (product/recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe (product/recruitment)
Hilary Turnbull
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com

IEEE Software
IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314
USA
Phone: +1 714 821 8380
Fax: +1 714 821 4010
www.computer.org
advertising@computer.org

References

1. C.F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Comm. ACM*, vol. 30, no. 5, 1987, pp. 416–429.
2. M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Trans. Software Eng.*, vol. 23, no. 11, 1997, pp. 736–743.
3. A. Miller, *Subset Selection in Regression*, 2nd ed., Chapman Hall, 2002.
4. M.A. Hall and G. Holmes, "Benchmarking Attribute Selection Techniques for Discrete Class Data Mining," *IEEE Trans. Knowledge and Data Eng.*, vol. 15, no. 6, 2003, pp. 1437–1447.
5. R. Kohavi, D. Sommerfield, and J. Dougherty, "Data Mining Using MLC++: A Machine Learning Library in C++," *Proc. 8th IEEE Int'l Conf. Tools with AI* (ICTAI 96), IEEE CS Press, 1996, pp. 234–235.
6. B. Boehm, *Software Engineering Economics*, Prentice Hall, 1981.
7. B. Boehm et al., *Software Cost Estimation with Cocomo II*, Prentice Hall, 2000.
8. S. Chulani, B. Boehm, and B. Steece, "Bayesian Analysis of Empirical Software Engineering Cost Models," *IEEE Trans. Software Eng.*, vol. 25, no. 4, 1999, pp. 573–583.
9. I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 1999.
10. T. Menzies et al., "Validation Methods for Calibrating Software Effort Models," *Proc. 27th Int'l Conf. Software Eng.* (ICSE 05), ACM Press, 2005, pp. 587–595; http://menzies.us/pdf/04coconut.pdf.
11. T. Menzies et al., "Simple Software Cost Estimation: Safe or Unsafe?" *Proc. 2005 Workshop Predictor Models in Software Eng.*, ACM Press, 2005; http://menzies.us/pdf/05safewhen.pdf.
12. J. Sayyad Shirabad and T.J. Menzies, *The Promise Repository of Software Engineering Databases*, School of Information Technology and Eng., Univ. of Ottawa, 2005; http://promise.site.uottawa.ca/SERepository.
13. Z. Chen, T. Menzies, and D. Port, "Feature Subset Selection Can Improve Software Cost Estimation Accuracy," *Proc. 2005 Workshop Predictor Models in Software Eng.*, ACM Press, 2005; http://menzies.us/pdf/05fsscocomo.pdf.