Feature Subset Selection Methods for COCOMO Based Software Effort Estimation

Daniel Baker,Tim Menzies *Member, IEEE*

***Abstract*— This paper demonstrates the results of feature subset selection methods for the COCOMO model.**

I. Introduction

IN 2005...(START WITH QUOTE AND REFERENCE ABOUT MONEY LOST FROM INACURRATE SOFTWARE COST ESTIMATION/BUDGETING) Currently cost estimation is accomplished using models such as COCOMO [1], (LIST MANY OTHER METHODS WITH REFERENCES). These models predict the development cost for a new software project based on past project data. For an accurate prediction the training data needs quantity, quality, and relevance to the new project(ADD REFERENCES). Unfortunately this is difficult in practice and estimates are often made using inadequate training data. Consequently, these models are plagued with problems including highly inaccurate predictions(FIND BEST REFERENCE) and the variance problem(INSERT REFERENCE). It has been shown that the variance can be reduced by feature subset selection methods that discard irrelevant, redundant, noisy, and unreliable attributes(LOTS OF REF HERE). This paper explores some of these attribute pruning techniques.

II. Background

A. COCOMO

The case study material for this paper uses COCOMO-format data. COCOMO (the COnstructive COst MOdel) was originally developed by Barry Boehm in 1981 [1] and was extensively revised in 2000 [3]. The core intuition behind COCOMO-based estimation is that as a program grows in size, the development effort grows exponentially. More specifically:

$$effort(personmonths) = a * \left(KLOC^b\right) * \left(\prod_j EM_j\right) \quad (1)$$

Here, $KLOC$ is thousands of delivered source instructions. $KLOC$ can be estimated directly or via a *function point estimation*. Function points are a product of five defined data components (inputs, outputs, inquiries, files, external interfaces) and 14 weighted environment characteristics (data comm, performance, reusability, etc.) [2], [3]. A 1,000 line Cobol program would typically implement about 14 function points, while a 1,000-line C program would implement about seven[1].

In Equation 1, EM_j is one of *effort multipliers* such as *cplx* (complexity) or *pcap* (programmer capability). In order to model the effects of EM_j on development effort, Boehm proposed reusing numeric values which he generated via regression on historical data for each value of EM_i.

In practice, effort data forms exponential distributions. Appendix B describes methods for using such distributions in effort modeling.

Note that in COCOMO 81, Boehm identified three common types of software: *embedded, semi-detached*, and *organic*. Each has their own characteristic "a" and "b" (see Figure 1). COCOMO-II ignores these distinctions. This study used data sets in both the COCOMO 81 and COCOMO-II format. For more on the differences between COCOMO 81 and COCOMO-II, see Appendix A.

B. Data

The software project data we used in this study came from two sources (see Figure 2). $Coc81$ is the original COCOMO data used by Boehm to calibrate COCOMO 81. $Nasa93$ comes from a NASA-wide database recorded in the COCOMO 81 format. This data has been in the public domain for several years but few have been aware of it. It can now be found on-line in several places including the PROMISE (Predictor Models in Software Engineering) web site[2]. $Nasa93$ was originally collected to create a NASA-tuned version of COCOMO, funded by the Space Station Freedom Program. $Nasa93$ contains data from six NASA centers including the Jet Propulsion Laboratory. Hence, it covers a very wide range of software domains, development processes, languages, and complexity as well as fundamental differences in culture and business practices between each center. All of these factors contribute to the large variances observed in this data set.

When the $nasa93$ data was collected, it was required that there be multiple interviewers with one person leading the interview and one or two others recording and checking documentation. Each data point was cross-checked with either official records or via independent subjective inputs from other project personnel who fulfilled various roles on the project. After the data was translated into the COCOMO 81 format, the data was reviewed with those who originally provided the data. Once sufficient data existed the data was analyzed to identify outliers and the data values were re-verified with

Dan Baker is with the Lane Department of Computer Science, West Virginia University and can be reached at danielryanbaker@gmail.com

Dr. Menzies is with the Lane Department of Computer Science, West Virginia University and can be reached at tim@menzies.us

This research was conducted with funds from the NASA Software Assurance Research Program led by the NASA IV&V Facility.

Manuscript received January 1, 2006; revised XXX YY, 200Z.

[1] http://www.qsm.com/FPGearing.html

[2] http://promise.site.uottawa.ca/SERepository/ and http://unbox.org/wisp/trunk/cocomo/data.

Mode	a	b	notes
Organic	3.2	1.05	projects from relatively small software teams develop software in a highly familiar, in-house environment.
Embedded	2.8	1.2	projects operating within (is embedded in) a strongly coupled complex of hardware, software, regulations, and operational procedures.
Semi-Detached	3.0	1.12	An intermediary mode between organic and embedded.

Fig. 1. Standard COCOMO 81 development modes.

the development teams once again if deemed necessary. This typically required from two to four trips to each NASA center. All of the supporting information was placed in binders, which we still on occasion reference even today.

Using Boehm's COCOMO-I "local calibration" the $nasa93$ data has been shown to contain large deviations due to the wide variety of projects in that data set, and *not* poor data collection (ADD REFERENCE TO COSEEKMO PAPER). Our belief is that $nasa93$ was collected using methods equal to, or better, than standard industrial practice. If so, then industrial data would suffer from deviations equal to or lager than those seen in the $nasa93$ data.

C. Performance Measures

The performance of models generating continuous output can be assessed in many ways, including PRED(30), MMRE, correlation, etc. PRED(30) is a measure calculated from the relative error, or RE, which is the relative size of the difference between the actual and estimated value. One way to view these measures is to say that training data contains records with variables $1, 2, 3, .., N$ and performance measures add additional new variables $N + 1, N + 2, ...$

The magnitude of the relative error, or MRE, is the absolute value of that relative error:

$$MRE = |predicted - actual|/actual$$

The mean magnitude of the relative error, or MMRE, is the average percentage of the absolute values of the relative errors over an entire data set. MMRE results were shown in Figure **??** in the *mean% average test error* column. Given T tests, MMRE is calculated as follows:

$$MMRE = \frac{100}{T}\sum_{i}^{T}\frac{|predicted_i - actual_i|}{actual_i}$$

PRED(N) reports the average percentage of estimates that were within N% of the actual values. Given T tests, then:

$$PRED(N) = \frac{100}{T}\sum_{i}^{T}\begin{cases} 1\ if\ MRE_i \leq \frac{N}{100} \\ 0\ otherwise \end{cases}$$

For example, a PRED(30)=50% means that half the estimates are within 30% of the actual.

Another performance measure of a model predicting numeric values is the correlation between predicted and actual values. Correlation ranges from +1 to -1 and a correlation of +1 means that there is a perfect positive linear relationship between variables. Appendix C shows how to calculate correlation.

All these performance measures (correlation, MMRE and PRED) address subtly different issues. Overall, PRED measures how *well* an effort model performs while MMRE measures *poor* performance. A single large mistake can skew the MMREs and not effect the PREDs. Shepperd and Schofield comment that:

> MMRE is fairly conservative with a bias against overestimates while PRED(30) will identify those prediction systems that are generally accurate but occasionally wildly inaccurate [10, p736].

Since they measure different aspects of model performance, COSEEKMO uses combinations of PRED, MMRE, and correlation (using the methods described later in this paper).

III. Attribute Ranking Driven Greedy Search Feature Subset Selection

FOLLOWING the slow but steady success of COSEEKMO, it becomes necessary to find a similarly accurate algorithm which is fast enough to be used by business users in the real world. We decided to begin the search for such an algorithm using a minimal approach, that is, a feature subset selection algorithm that used a greedy search to minimize evaluations of the attribute space. We decided to rank the attributes using correlation and to use the ranked results as the order to grow the attribute set in the case of a forward select greedy search, or prune the set in the case of a backward elimination greedy search. At each step the attribute set is evaluated with the MMRE, Pred(30), deviation, and correlation from using COCOMO regression with the attribute set in question. If the change is considered an improvement then it is kept and the search continues, otherwise it stops. We introduced a horizon variable that could be set to allow for the search to continue even without improvement. An extensive experiment was run to benchmark the many customizations of the greedy FSS approach against standard COCOMO least squares regression as described in the pseudocode in Figure 3. The results are shown in Figure 14 and Figure 15. For the COC81 dataset, standard COCOMO least squares regression was found to be far superior. However, for the NASA93 dataset, many customizations of the greedy feature subset selection had similar or slighty better MMRE and Pred(30) values, and lower deviations. (refer to background on datasets to explain this).

Although quick, this algorithm is not accurate enough to be an acceptable replacement for even avoiding feature subset selection entirely. Although it had some slight success with the NASA93 dataset it failed to provide similar results to COCOMO regression on the COC81 dataset. Noticing that the results tended to improve with higher horizon values, we reasoned that the attribute space needed more exploration. Finally, with a dataset as small as the COC81 and NASA93 datasets, the importance of heuristics to prune the state space explosion were examined. Next we decided to build an efficient algorithm that evaluated all 32,768 attribute combinations.

Data sources	*Coc81:*63 records in the COCOMO 81 format. Source: [1, p496-497]. Download from `http://unbox.org/wisp/trunk/cocomo/data/coc81modeTypeLangType.csv`. *Nasa93:* 93 NASA records in the COCOMO 81 format. Download from `http://unbox.org/wisp/trunk/cocomo/data/nasa93.csv`. *CocII:* 161 records in the COCOMO II format from the COCOMO consortium (co-ordinated by USC). This data is not in the public domain.
Data subsets	*All:* selects all records from a particular source; e.g. "coc81_all". *Category:* is a NASA-specific designation selecting the type of project; e.g. avionics, data capture, etc. *Dev:* indicates the development methodology; e.g. div.waterfall. *DevEnd:* shows the last year of the software project. *Fg:* selects either "f" (flight) or "g" (ground) software. *Kind:* selects records relating to the development platform; max= mainframe and mic= microprocessor. *Lang:* selects records about different development languages. *Project* and *center*: $nasa93$ designations selecting records relating to where the software was built and the name of the project. *Mode=e:* selects records relating to the *embedded* COCOMO 81 development mode. The different COCOMO 81 development models were described in Figure 1. *Mode=o:* selects COCOMO 81 *organic* mode records. *Mode=sd:* selects COCOMO 81 *semi-detached* mode records. *Org:* is a $cocII$ designation showing what organization provided the data. *Size:* is a $cocII$ specific designation grouping the records into (e.g.) those around 100KLOC. *Type:* selects different $coc81$ designations and include "bus" (for business application) or "sys" (for system software). *Year:* is a $nasa93$ term that selects the development years, grouped into units of five; e.g. 1970,1971,1972,1973,1974 are labeled "1970".

Fig. 2. Data sets (top) and parts (bottom) of the data used in this study.

IV. COCOMOST

```
for data in dataSets
  for i in 1 to 30
    test = randomRecords(data,10)
    train = data - test
    attributes = (all COCOMO 81 attributes)
    results = LC(test, train, attributes)
    print variables and results
    for attribute in COCOMO_81_attributes
      rank correlation using LC(train, train, attribute)
      rankedList += sorted list of ranked attributes
    for search in "forward backward"
      for horizon in "0 1 2 4 8 16"
        for eval in "mmre sd_mre pred30 correlation"
          attr(test, train, search, horizon, eval, rankedList)

attr()
  for attribute in rankedList
    newSet = bestSet + attribute
    if (search==backward)
      tmpSet=inverse(newSet)
    else
      tmpSet = newSet
    oldScore = newScore
    results = LC(train, train, tmpSet)
    newScore = results.eval
    if (newScore better than oldScore)
      bestSet = newSet
      stale = horizon
    else
      newScore = oldScore
      stale--
    if (stale < 1)
      exit for
  next attribute
  if (search==backward)
    bestSet=inverse(bestSet)
  finalResults = LC(test, train, bestSet)
  print variables and results
```

Fig. 3. This pseudocode outlines an experiment that compared standard COCOMO 81 local calibration with an approach that ranks the attributes based on correlation, and then builds a subset of attributes using a greedy search guided by the attribute ranking.

THE COCOMOST algorithm, as outlined in Figure 5, uses feature subset selection to prune irrelevant, redundant, noisy, and unreliable attributes from the COCOMO model. It executes a complete search of the attribute space, evaluating attribute sets using local calibration. Thus, it is a "wrapper" attribute selection technique instead of a "filter" because it evaluates using the target learner [5]. However, it shares several of the advantages of a filter. Unlike most wrappers, COCOMOST is fast enough to search the entire attribute space instead of using heuristics to limit the state space explosion. This introduces the vulnerability that every attribute beyond

```
for data in dataSets
  for i in 1 to 30
    test       = randomRecord(data)
    train      = data - test
    attributes = (all COCOMO 81 attributes)
    results    = LC(test, train, attributes)
    print variables and results
    attributes = cocomost(train)
    results    = LC(test, train, attributes)
    print variables and results
```

Fig. 4. This experiment benchmarks standard COCOMO-based local calibration against local calibration that uses cocomost to perform feature subset selection.

```
attributes = null
bestMMRE = LC(train, attributes)
bestAttributes = null
for attributes in 2^15
  newMMRE = LC(train, attributes)
  if newMMRE > bestMMRE
      bestMMRE = newMMRE
      bestAttributes = attributes

return bestAttributes
```

Fig. 5. Cocomost performs a complete search over the attribute space and evaluates the attribute sets using the target learner: COCOMO-based local calibration.

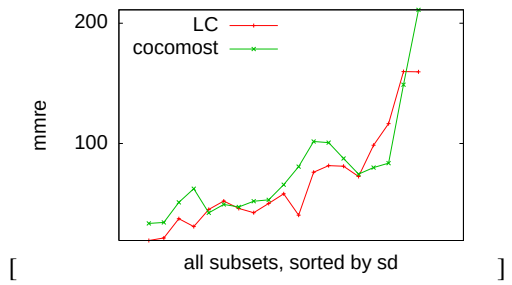

Fig. 6. Mean Magnitude of Relative Error

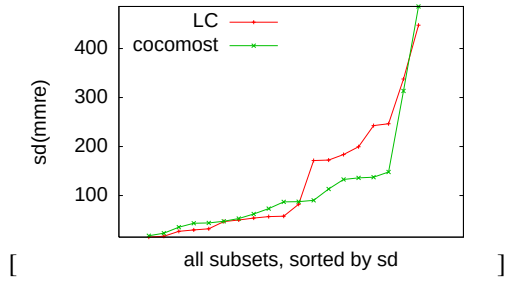

Fig. 7. Standard Deviation of Mean Magnitude of Relative Error

the 15 used in this study will double COCOMOST's execution time.

To compare the effectiveness of standard local calibration versus feature subset selection with COCOMOST followed by local calibration, we randomly pulled a test set from the training data 30 times and made estimates with each learner as described in Figure 4.

V. Results

(PULL QUARTILE CHARTS TEXT FROM MENZIES PAPER)

Figure 6 shows the mean magnitude of relative error for each of the 19 subsets of data. Standard local calibration is shown in red and COCOMOST is in green. The results show similar errors between the learners. Figure 7 is similarly structured, except it displays the standard deviation of the error. These results show that COCOMOST greatly reduces the variance for a large portion of the datasets.

VI. Conclusion

THE results show the limits of feature subset selection using local calibration as the evaluation criteria. COCOMOST produced statistically equivalent errors as Boehm's COCOMO model. Since the attribute space was searched completely, this indicates that more involved evaluation methods or models need to be explored to reduce the error. However, COCOMOST was able to reduce the variance in the model which will make it easier to distinguish rival methods. This will provide a stepping stone for further research. In addition, for this evaluation method, heuristics were found to be less effective and slower than a finely tuned complete search of the attribute space.

VII. Future Work

THERE is a lot of work to be done in improving software cost estimation models. Some possibilities suggested by this research include:

- Feature Subset Selection using other evaluation methods and learners.
- Expanding upon COCOMOST with more data mining techniques such as bagging(ADD REFERENCE).
- More comparisons of learners using COCOMOST as a "strawman" FSS model.
- Due to COCOMOST's fairly quick runtime it could be used by WRAPPER algorithms (add ref?) such as COSEEKMO.

References

[1] B. Boehm. *Software Engineering Economics*. Prentice Hall, 1981.
[2] T. Jones, *Estimating Software Costs*. McGraw-Hill, 1998.
[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] I. H. Witten and E. Frank, *Data mining. 2nd edition*. Los Altos, US: Morgan Kaufmann, 2005.
[5] M. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions On Knowledge And Data Engineering*, vol. 15, no. 6, pp. 1437–1447, 2003.
[6] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Feature subset selection can improve software cost estimation accuracy." in *Proceedings of the 2005 Workshop on Predictor Models in Software Engineering*, ACM Press, New York, NY, May 2005.
[7] T. Menzies, D. Port, Zhihao Chen, and J. Hihn, "Validation methods for calibrating software effort models," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on* vol., no.pp. 587- 595, 15–21 May 2005.
[8] T. Menzies and J. Hihn, "Evidence-Based Cost Estimation for Better-Quality Software," in *IEEE Software* , vol.23, no.4, pp. 64–66, July-Aug. 2006.
[9] G. V. Lashkia and L. Anthony, "Relevant, irredundant feature selection and noisy example elimination," in *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol.34, no.2, pp. 888–897, 2004.
[10] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 12, November 1997, available from http://www.utdallas.edu/~rbanker/SE_XII.pdf.
[11] K. Strike, K. E. Emam and N. H. Madhavji, "Software Cost Estimation with Incomplete Data" in *IEEE Transactions on Software Engineering*, vol.27, no.10, pp. 890–908, 2001.
[12] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

Tim Menzies is an associate professor at the University of West Virginia 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 AI and knowledge engineering from the University of New South Wales. Contact him at tim@menzies.us.

Dan Baker is a graduate student at West Virginia University. He received his Bachelor's degree in Computer Science from West Virginia University in May of 2006. For his senior project he developed the database used in the West Virginia Crime Reduction and Information Management Effort. He is currently working on his Master's degree in Computer Science at West Virginia University while working as a research assistant for Dr Menzies. His research focuses on data mining in data starved domains such as software cost estimation. He is the president of the student chapter of Upsilon Pi Epsilon at West Virginia University. UPE is the first and only international honor society for the Computing Sciences.

APPENDIX

APPENDIX A - COCOMO-I vs COCOMO-II: In COCOMO II, the exponential COCOMO 81 term b was expanded into the following expression:

$$b + 0.01 * \sum_j SF_j \quad (2)$$

where b is 0.91 in COCOMO II 2000, and SF_j is one of five *scale factors* that exponentially influence effort. Other changes in COCOMO II included dropping the development modes of Figure 1 as well as some modifications to the list of effort multipliers and their associated numeric constants (see appendix E).

APPENDIX B - Calculating Correlation: Given a test set of size T, correlation is calculated as follows:

$$\begin{array}{lll} \bar{p} & = \frac{\sum_I^T predicted_i}{T} & \bar{a} = \frac{\sum_I^T actual_i}{T} \\ S_p & = \frac{\sum_i^T (predicted_i - \bar{p})^2}{T-1} & S_a = \frac{\sum_i^T (actual_i - \bar{a})^2}{T-1} \\ S_{pa} & = \frac{\sum_i^T (predicted_i - \bar{p})(actual_i - \bar{a})}{T-1} & \\ corr & = S_{pa} / \sqrt{S_p * S_a} & \end{array}$$

APPENDIX C - Local Calibration: (CHANGE TO METHOD WITHOUT USING LOGGED NUMERICS) This approach assumes that a matrix $D_{i,j}$ holds:

- The natural log of the $KLOC$ estimates;
- The natural log of the actual efforts for projects $i \leq$j$\leq t$;
- The natural logarithm of the cost drivers (the scale factors and effort multipliers) at locations $1 \leq i \leq 15$ (for COCOMO 81) or $1 \leq i \leq 22$ (for COCOMO-II).

With those assumptions, Boehm [1] shows that for COCOMO 81, the following calculation yields estimates for "a" and "b" that minimizes the sum of the squares of residual errors:

$$\left.\begin{array}{rcl} EAF_i & = & \sum_j^N D_{i,j} \\ a_0 & = & t \\ a_1 & = & \sum_i^t KLOC_i \\ a_2 & = & \sum_i^t (KLOC_i)^2 \\ d_0 & = & \sum_i^t (actual_i - EAF_i) \\ d_1 & = & \sum_i^t ((actual_i - EAF_i) * KLOC_i) \\ b & = & (a_0 d_1 - a_1 * d_0)/(a_0 a_2 - a_1^2) \\ a_3 & = & (a_2 d_0 - a_1 d_1)/(a_0 a_2 - a_1^2) \\ a & = & e^{a_3} \end{array}\right\} \quad (3)$$

APPENDIX D - COCOMO Numerics: Figure 8 shows the COCOMO 81 EM_j (effort multipliers). The effects of that multiplier on the effort are shown in Figure 9. Increasing the *upper* and *lower* groups of variables will *decrease* or *increase* the effort estimate, respectively.

upper: increase these to decrease effort	acap: analysts capability pcap: programmers capability aexp: application experience modp: modern programming practices tool: use of software tools vexp: virtual machine experience lexp: language experience
middle	sced: schedule constraint
lower: decrease these to increase effort	data: data base size turn: turnaround time virt: machine volatility stor: main memory constraint time: time constraint for cpu rely: required software reliability cplx: process complexity

Fig. 8. COCOMO 81 effort multipliers.

		very low	low	nominal	high	very high	extra high
upper (increase these to decrease effort)	ACAP	1.46	1.19	1.00	0.86	0.71	
	PCAP	1.42	1.17	1.00	0.86	0.70	
	AEXP	1.29	1.13	1.00	0.91	0.82	
	MODP	1.2	1.10	1.00	0.91	0.82	
	TOOL	1.24	1.10	1.00	0.91	0.83	
	VEXP	1.21	1.10	1.00	0.90		
	LEXP	1.14	1.07	1.00	0.95		
middle	SCED	1.23	1.08	1.00	1.04	1.10	
lower (increase these to increase effort)	DATA		0.94	1.00	1.08	1.16	
	TURN		0.87	1.00	1.07	1.15	
	VIRT		0.87	1.00	1.15	1.30	
	STOR			1.00	1.06	1.21	1.56
	TIME			1.00	1.11	1.30	1.66
	RELY	0.75	0.88	1.00	1.15	1.40	
	CPLX	0.70	0.85	1.00	1.15	1.30	1.65

Fig. 9. The precise COCOMO 81 effort multiplier values.

Figure 10 shows the COCOMO 81 effort multipliers of Figure 9, rounded and simplified to two significant figures.

Figure 11, Figure 12 and Figure 13 show the COCOMO-II values analogies to Figure 8, Figure 9 and Figure 10 (respectively).

APPENDIX E:

		very low	low	nominal	high	very high	extra high
upper (increase these to decrease effort)	ACAP	1.2	1.1	1.00	0.9	0.8	
	PCAP	1.2	1.1	1.00	0.9	0.8	
	AEXP	1.2	1.1	1.00	0.9	0.8	
	MODP	1.2	1.1	1.00	0.9	0.8	
	TOOL	1.2	1.1	1.00	0.9	0.8	
	VEXP	1.2	1.1	1.00	0.9		
	LEXP	1.2	1.1	1.00	0.9		
middle	SCED	1.2	1.1	1.00	1.1	1.2	
lower (increase these to increase effort)	DATA		0.9	1.00	1.1	1.2	
	TURN		0.9	1.00	1.1	1.2	
	VIRT		0.9	1.00	1.1	1.2	
	STOR			1.00	1.1	1.2	1.3
	TIME			1.00	1.1	1.2	1.3
	RELY	0.8	0.9	1.00	1.1	1.2	
	CPLX	0.8	0.9	1.00	1.1	1.2	1.3

Fig. 10. Rounded COCOMO 81 effort multiplier values.

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

Fig. 11. The COCOMO II scale factors and effort multipliers.

		extra low	very low	low	nominal	high	very high	extra high
scale	prec		6.20	4.96	3.72	2.48	1.24	0.00
factors	flex		5.07	4.05	3.04	2.03	1.01	0.00
(exponentially	resl		7.07	5.65	4.24	2.83	1.41	0.00
decreases	team		5.48	4.38	3.29	2.19	1.10	0.00
effort)	pmat		7.80	6.24	4.68	3.12	1.56	0.00
upper	acap		1.42	1.19	1.00	0.85	0.71	
(linearly	pcap		1.34	1.15	1.00	0.88	0.76	
decreases	pcon		1.29	1.12	1.00	0.90	0.81	
effort)	aexp		1.22	1.10	1.00	0.88	0.81	
	pexp		1.19	1.09	1.00	0.91	0.85	
	ltex		1.20	1.09	1.00	0.91	0.84	
	tool		1.17	1.09	1.00	0.90	0.78	
	site		1.22	1.09	1.00	0.93	0.86	0.80
	sced		1.43	1.14	1.00	1.00	1.00	
lower	rely		0.82	0.92	1.00	1.10	1.26	
(linearly	data			0.90	1.00	1.14	1.28	
increases	cplx		0.73	0.87	1.00	1.17	1.34	1.74
effort)	ruse			0.95	1.00	1.07	1.15	1.24
	docu		0.81	0.91	1.00	1.11	1.23	
	time				1.00	1.11	1.29	1.63
	stor				1.00	1.05	1.17	1.46
	pvol			0.87	1.00	1.15	1.30	

Fig. 12. The precise COCOMO II numerics.

		extra low	very low	low	nominal	high	very high	extra high
Scale	PREC		6.3	5.1	3.8	2.5	1.3	0
Factors	FLEX		6.3	5.1	3.8	2.5	1.3	0
	RESL		6.3	5.1	3.8	2.5	1.3	0
	TEAM		6.3	5.1	3.8	2.5	1.3	0
	PMAT		6.3	5.1	3.8	2.5	1.3	0
upper	ACAP		1.3	1.1	1.0	0.9	0.8	
	PCAP		1.3	1.1	1.0	0.9	0.8	
	PCON		1.3	1.1	1.0	0.9	0.8	
	AEXP		1.3	1.1	1.0	0.9	0.8	
	PEXP		1.3	1.1	1.0	0.9	0.8	
	LTEX		1.3	1.1	1.0	0.9	0.8	
	TOOL		1.3	1.1	1.0	0.9	0.8	
	SITE		1.3	1.1	1.0	0.9	0.8	0.8
	SCED		1.3	1.1	1.0	0.9	0.8	
lower	RELY		0.8	0.9	1.0	1.1	1.3	
	DATA			0.9	1.0	1.1	1.3	
	CPLX		0.8	0.9	1.0	1.1	1.3	1.5
	RUSE			0.9	1.0	1.1	1.3	1.5
	DOCU		0.8	0.9	1.0	1.1	1.3	
	TIME				1.0	1.1	1.3	1.5
	STOR				1.0	1.1	1.3	1.5
	PVOL			0.9	1.0	1.1	1.3	

Fig. 13. The rounded COCOMO II numerics.

Data	Learn	Search	Hrzn	Eval	Avg MMRE	Avg SD	Avg Pred30
coc81	LC	N/A	N/A	N/A	44.72	37.28	38.75
coc81	attr	forward	0	mmre	93.63	101.68	27.92
coc81	attr	forward	0	sd_mre	93.63	101.68	27.92
coc81	attr	forward	0	pred30	93.63	101.68	27.92
coc81	attr	forward	0	corr	93.63	101.68	27.92
coc81	attr	forward	1	mmre	49.42	42.23	36.25
coc81	attr	forward	1	sd_mre	65.11	59.69	32.50
coc81	attr	forward	1	pred30	83.87	81.42	26.67
coc81	attr	forward	1	corr	64.39	58.99	30.00
coc81	attr	forward	2	mmre	45.73	37.66	38.75
coc81	attr	forward	2	sd_mre	51.27	45.17	37.08
coc81	attr	forward	2	pred30	76.11	70.23	26.67
coc81	attr	forward	2	corr	57.77	50.99	32.50
coc81	attr	forward	4	mmre	45.73	37.66	38.75
coc81	attr	forward	4	sd_mre	49.08	42.47	37.92
coc81	attr	forward	4	pred30	72.18	67.43	28.75
coc81	attr	forward	4	corr	53.95	46.20	33.75
coc81	attr	forward	8	mmre	45.73	37.66	38.75
coc81	attr	forward	8	sd_mre	49.08	42.47	37.92
coc81	attr	forward	8	pred30	69.70	63.81	27.50
coc81	attr	forward	8	corr	53.95	46.20	33.75
coc81	attr	forward	16	mmre	45.73	37.66	38.75
coc81	attr	forward	16	sd_mre	49.08	42.47	37.92
coc81	attr	forward	16	pred30	69.53	63.22	27.08
coc81	attr	forward	16	corr	53.95	46.20	33.75
coc81	attr	back	0	mmre	49.99	43.76	35.00
coc81	attr	back	0	sd_mre	49.99	43.76	35.00
coc81	attr	back	0	pred30	49.99	43.76	35.00
coc81	attr	back	0	corr	49.99	43.76	35.00
coc81	attr	back	1	mmre	49.99	43.76	35.00
coc81	attr	back	1	sd_mre	50.32	44.34	35.00
coc81	attr	back	1	pred30	52.50	45.07	34.17
coc81	attr	back	1	corr	51.20	45.19	36.25
coc81	attr	back	2	mmre	50.18	43.67	35.00
coc81	attr	back	2	sd_mre	50.41	44.64	35.00
coc81	attr	back	2	pred30	54.01	45.74	32.92
coc81	attr	back	2	corr	53.51	46.26	32.50
coc81	attr	back	4	mmre	51.47	44.53	34.17
coc81	attr	back	4	sd_mre	52.70	48.03	33.75
coc81	attr	back	4	pred30	55.97	47.29	31.67
coc81	attr	back	4	corr	54.05	46.27	32.50
coc81	attr	back	8	mmre	52.43	45.89	33.75
coc81	attr	back	8	sd_mre	54.46	49.98	35.42
coc81	attr	back	8	pred30	58.27	49.29	27.50
coc81	attr	back	8	corr	54.31	46.56	32.08
coc81	attr	back	16	mmre	52.43	45.89	33.75
coc81	attr	back	16	sd_mre	54.66	50.33	35.83
coc81	attr	back	16	pred30	59.70	50.37	27.50
coc81	attr	back	16	corr	54.31	46.56	32.08

Fig. 14. Standard COCOMO Local Calibration vs. Greedy FSS for the COC81 dataset.

APPENDIX F:

Data	Learn	Search	Hrzn	Eval	Avg MMRE	Avg SD	Avg Pred30
nasa93	LC	N/A	N/A	N/A	43.95	49.96	52.92
nasa93	attr	forward	0	mmre	52.88	54.98	47.08
nasa93	attr	forward	0	sd_mre	52.88	54.98	47.08
nasa93	attr	forward	0	pred30	52.88	54.98	47.08
nasa93	attr	forward	0	corr	52.88	54.98	47.08
nasa93	attr	forward	1	mmre	45.94	46.05	49.17
nasa93	attr	forward	1	sd_mre	48.45	50.57	50.42
nasa93	attr	forward	1	pred30	48.77	49.74	48.33
nasa93	attr	forward	1	corr	44.16	43.57	47.50
nasa93	attr	forward	2	mmre	44.21	44.06	50.00
nasa93	attr	forward	2	sd_mre	48.45	50.19	49.17
nasa93	attr	forward	2	pred30	46.20	45.71	47.08
nasa93	attr	forward	2	corr	43.23	43.17	51.25
nasa93	attr	forward	4	mmre	44.51	44.06	50.42
nasa93	attr	forward	4	sd_mre	47.22	50.29	50.00
nasa93	attr	forward	4	pred30	46.73	46.90	48.75
nasa93	attr	forward	4	corr	41.95	42.06	53.75
nasa93	attr	forward	8	mmre	44.05	44.23	51.67
nasa93	attr	forward	8	sd_mre	46.88	49.17	50.00
nasa93	attr	forward	8	pred30	46.02	47.31	50.83
nasa93	attr	forward	8	corr	41.72	41.72	53.33
nasa93	attr	forward	16	mmre	44.05	44.23	51.67
nasa93	attr	forward	16	sd_mre	46.82	49.42	49.58
nasa93	attr	forward	16	pred30	46.02	47.31	50.83
nasa93	attr	forward	16	corr	41.72	41.72	53.33
nasa93	attr	back	0	mmre	42.34	43.50	52.08
nasa93	attr	back	0	sd_mre	42.34	43.50	52.08
nasa93	attr	back	0	pred30	42.34	43.50	52.08
nasa93	attr	back	0	corr	42.34	43.50	52.08
nasa93	attr	back	1	mmre	41.66	41.23	52.50
nasa93	attr	back	1	sd_mre	41.77	43.19	52.08
nasa93	attr	back	1	pred30	42.45	42.11	52.92
nasa93	attr	back	1	corr	42.85	41.65	50.00
nasa93	attr	back	2	mmre	44.76	44.95	50.00
nasa93	attr	back	2	sd_mre	48.67	51.18	48.75
nasa93	attr	back	2	pred30	41.93	41.19	51.25
nasa93	attr	back	2	corr	42.23	43.24	50.83
nasa93	attr	back	4	mmre	45.72	46.65	52.08
nasa93	attr	back	4	sd_mre	47.60	48.90	49.17
nasa93	attr	back	4	pred30	42.79	43.44	50.83
nasa93	attr	back	4	corr	41.87	42.37	51.25
nasa93	attr	back	8	mmre	45.77	47.00	54.58
nasa93	attr	back	8	sd_mre	47.60	48.90	49.17
nasa93	attr	back	8	pred30	44.52	45.91	49.17
nasa93	attr	back	8	corr	41.91	42.59	52.08
nasa93	attr	back	16	mmre	45.77	47.00	54.58
nasa93	attr	back	16	sd_mre	47.60	48.90	49.17
nasa93	attr	back	16	pred30	44.78	46.42	48.75
nasa93	attr	back	16	corr	41.91	42.59	52.08

Fig. 15. Standard COCOMO Local Calibration vs. Greedy FSS for the NASA93 dataset.