A Hybrid Approach to Expert and Model Based Effort Estimation

Daniel Ryan Baker

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

Lane Department of Computer Science and Electrical Engineering
Morgantown, West Virginia
2007

Categories and Subject Descriptors:
D.2.9 Cost estimation, I.2.3 Uncertainty, "fuzzy," and probabilistic reasoning,
H.4.2 Decision support, I.2.6 Learning, I.5.2 Feature evaluation and selection,
I.6.8 Monte Carlo, K.6.3 software process

Keywords: COCOMO, uncertainty, instance selection, feature selection, boosting, bagging

Abstract

A Hybrid Approach to
Expert and Model Based
Effort Estimation

Daniel Ryan Baker

It is important to have a good cost estimate in order to budget a new project. Unfortunately, software effort estimation methods are often inaccurate. Molokken and Jorgensen report that 60-80% of the time a software project will overrun its estimate by an average of 30% [98]. Furthermore, most estimates do not describe the uncertainty of the estimate [52,54,118]. In addition, each source of uncertainty, as described by Kitchenham and Linkman [68], has yet to be represented.

In this thesis, the design principles of an effort estimation tool called 2CEE are discussed. This tool is currently being deployed at NASA's Jet Propulsion Laboratory, and feedback regarding the methodology improvements to industry is reported. This tool represents an approach to effort estimation that provides greater interaction of the cost analyst with the estimation model. The approach places an emphasis on representing estimation uncertainty which, among other benefits, allows estimates with increasing confidence throughout the software lifecycle.

Previously, Jorgenson has argued that most effort estimation is done manually [49, 52]. However, manual methods have difficulty sampling the space of uncertainty [54]. This thesis describes how 11 of Jorgensen's 12 expert judgment best practices may be automated in a model, 7 of which are demonstrated in 2CEE. Thus, the distinction between manual and automated methods for effort estimation is questioned. Instead, an alternate ideology is proposed where neither manual nor automatic estimation methods dominate, but rather each augments the other.

In addition, the techniques of feature subset selection, bagging, and boosting are explored for the COCOMO software effort estimation model. These methods are evaluated using nonparametric techniques to compensate for the non-Gaussian error distributions [95]. Improved estimation accuracy is reported. Finally, a startling discovery regarding the stability of the COCOMO software effort estimation model is reported.

Contents

1 Introduction **1**
- 1.1 Contribution of This Thesis . . . 7
- 1.2 Structure of This Document . . . 7

2 Related Work **9**
- 2.1 Software Effort Estimation Taxonomy . . . 9
- 2.2 Sparse Data Methods for Software Effort Estimation . . . 9
  - 2.2.1 Expert Judgment . . . 9
  - 2.2.2 Analytic Hierarchy Process . . . 12
  - 2.2.3 Case Based Reasoning . . . 13
- 2.3 Many Data Methods for Software Effort Estimation . . . 15
  - 2.3.1 Arbitrary Function Approximators . . . 15
  - 2.3.2 Function Based Software Effort Estimation . . . 17
- 2.4 Software Effort Estimation Best Practices . . . 25
- 2.5 Open Issues in Software Effort Estimation . . . 25
  - 2.5.1 Inaccuracy . . . 25
  - 2.5.2 Scarce Data . . . 27
  - 2.5.3 Method Evaluation . . . 28
  - 2.5.4 Uncertainty . . . 33
  - 2.5.5 The ”Research to Reality” Divide . . . 36
  - 2.5.6 Overly Optimistic Estimation . . . 37
- 2.6 Feature Selection . . . 38
  - 2.6.1 Filters . . . 39
  - 2.6.2 Wrappers . . . 39
  - 2.6.3 Feature Space . . . 40
  - 2.6.4 Feature Selection for Software Effort Estimation . . . 40
- 2.7 Record Selection . . . 41
  - 2.7.1 Manual Record Selection . . . 42
  - 2.7.2 Automatic Record Selection . . . 42
- 2.8 Meta-learners . . . 43
  - 2.8.1 Stacking . . . 43
  - 2.8.2 Bagging . . . 44
  - 2.8.3 Boosting . . . 45
- 2.9 Summary of Related Work . . . 47

3 Laboratory Studies **49**
3.1 Experimental Design 49
3.1.1 Data Used in This Study 50
3.1.2 Method Evaluation 52
3.2 Feature Subset Selection 53
3.2.1 Cocomost 53
3.2.2 dBFS 56
3.2.3 Cocomin 58
3.2.4 Comparing Each FSS Method 60
3.3 Bagging 62
3.3.1 1st Bagging Experiment 62
3.3.2 2nd Bagging Experiment 64
3.3.3 3rd Bagging Experiment: Cocomonster 66
3.4 Boosting 67
3.4.1 Boosting by Subsampling 67
3.4.2 Boosting by Oversampling 69
3.4.3 Distributive Boosting with Adaboost 71
3.5 Summary of Experimental Results 73

4 The Methodology of 2CEE **76**
4.1 Introduction 76
4.2 Combining Aspects of Experts and Models 76
4.3 Model Calibration Approach in 2CEE 77
4.4 Managing Uncertainty 78
4.4.1 Addressing Measurement and Assumption Error 78
4.4.2 Addressing Model Error 80
4.4.3 Addressing Scope Error 80
4.5 Evaluation Tools 81

5 Industrial Studies **82**
5.1 Implemented Expert Estimation Best Practices 82
5.2 An Industrial Implementation of the 2CEE Methodology 83
5.3 Industrial Evaluation of 2CEE Application 92
5.4 Unexpected Results 96
5.5 Feedback from Industry 97

6 Conclusion **101**
6.1 Future Work 108
6.1.1 Stacking Estimation Methods 108
6.1.2 Implement More Best Practices in 2CEE 109
6.1.3 Include More Models in 2CEE 110
6.1.4 Interactive Feature Exploration 110
6.1.5 Tradeoff Analysis 111

List of Figures

2.1 Myrtveit's taxonomy of software effort estimation methods. [106] 10
2.2 Standard COCOMO 81 development modes. 20
2.3 The COCOMO II scale factors and effort multipliers. 21
2.4 The precise COCOMO II numerics. 22
2.5 Example of SCAT's probabilistic output. [80] 24
2.6 Three different categories of effort estimation best practices: expert-based (at top); model-based (in the middle); methods that combine expert and model-based (at bottom). 26
2.7 Mann-Whitney U Test Example 31
2.8 The Original Adaboost Algorithm [32] 46

5.1 2CEE Historical Data Screenshot 85
5.2 2CEE Stratification Data Screenshot 86
5.3 2CEE Project Data Screenshot 87
5.4 2CEE Calibration View Screenshot 88
5.5 2CEE Cross Validation Screenshot 89
5.6 2CEE Estimate Screenshot 90
5.7 2CEE Options Screenshot 91
5.8 2CEE Batch Experimenter Screenshot 93
5.9 2CEE vs Standard Methods 96
5.10 COCOMO II coefficients using the NASA07 dataset 97
5.11 COCOMO coefficients using the NASA93 dataset 98
5.12 COCOMO coefficients using the COC81 dataset 99

6.1 2CEE vs Standard Methods 106

List of Tables

2.1 COCOMO Features 19
2.2 COCOMO Effort Multiplier Numerics 19

3.1 Cocomost FSS Experiment Results from All Data 54
3.2 Cocomost FSS Experiment Results from Coc81 Data 55
3.3 Cocomost FSS Experiment Results from Nasa93 Data 55
3.4 Selected dBFS Results from All Data 57
3.5 Selected dBFS Results from Coc81 Data 57
3.6 Selected dBFS Results from Nasa93 Data 58
3.7 Selected Cocomin Results from All Data 60
3.8 Selected Cocomin Results from Coc81 Data 61
3.9 Selected Cocomin Results from Nasa93 Data 61
3.10 Experiment Comparing FSS Methods on All Data 62
3.11 Experiment Comparing FSS Methods on Coc81 Data 62
3.12 Experiment Comparing FSS Methods on Nasa93 Data 62
3.13 Selected Results from 1st Bagging Experiment on All Data) 63
3.14 Selected Results from 1st Bagging Experiment on Coc81 Data) 64
3.15 Selected Results from 1st Bagging Experiment on Nasa93 Data) 64
3.16 Selected Results from 2nd Bagging Experiment on All Data) 65
3.17 Selected Results from 2nd Bagging Experiment on Coc81 Data) 65
3.18 Selected Results from 2nd Bagging Experiment on Nasa93 Data) 65
3.19 Cocomonster Bagging Results from All Data 66
3.20 Cocomonster Bagging Results from Coc81 Data 66
3.21 Cocomonster Bagging Results from Nasa93 Data 66
3.22 Selected Boosting by Subsampling Results on All Data) 68
3.23 Selected Boosting by Subsampling Results on Coc81 Data) 68
3.24 Selected Boosting by Subsampling Results on Nasa93 Data) 68
3.25 1st Boosting by Oversampling Experiment Results on All Data 69
3.26 Selected Boosting by Oversampling 2nd Results on All Data) 70
3.27 Selected Boosting by Oversampling 3rd Results on All Data) 70
3.28 Selected Boosting by Oversampling 4th Results on All Data) 71
3.29 Selected Adaboost 1st Results on All Data) 72
3.30 Selected Adaboost 1st Results on Coc81 Data) 72
3.31 Selected Adaboost 1st Results on Nasa93 Data) 73
3.32 Results from 2nd Adaboost Experiment on All Data 73
3.33 Results from 2nd Adaboost Experiment on Coc81 Data 74

3.34 Results from 2nd Adaboost Experiment on Nasa93 Data 74

5.1 Selected Results from JPL Experiment on All Data 94
5.2 Selected Results from JPL Experiment on Flight Data 94
5.3 Selected Results from JPL Experiment on Ground Data 95

Acknowledgments

Thank you West Virginia University for the opportunity to study computer science. I am very grateful to my research advisor Tim Menzies, our correspondents at JPL, Jairus Hihn and Karen Lum, and fellow graduate student Omid Djalali for their guidance, support, and criticism during my time as a graduate research assistant. Tim was a great help getting started in the research and bringing me back to the center when I became sidetracked. He was always there to review and comment on my progress whether it be coding experiments or writing papers. I am very grateful to Tim for teaching me his programming style. Who knew scripting was so powerful? I am especially thankful to Tim, Jairus, and Karen for the internship at NASA JPL, as well as for going to the HICSS conference. The internship was a great success thanks to the daily interaction with Jairus and Karen. I particularly want to thank Jairus for his help with regression, and to Karen for testing the tool. Thanks to Omid for your evaluation research, writing LOCOMO, helping me with bugs, and for driving me to get groceries after I wrecked my car.

I am grateful for my professors at West Virginia University, especially Tim Menzies for his data mining course, Arun Ross for teaching pattern recognition and machine learning, John Atkins for his painfully effective database courses, Jim Mooney for his computer architecture courses, and Elaine Eschen for algorithms. Thanks to Tim Menzies, Jairus Hihn, and Afzel Noore for serving on my committee.

I am thankful to the my fellow students for all of the conversations, advice, humor, and comradery. When times get tough, it's good to know you're not alone. I especially want to thank Omid Djalali, Oussama Elrawas, Nathan Moore, Philip Green, Zach Milton, Brian Sowers, and DJ Boland.

Finally, I am grateful to my parents and family for their moral support.

Chapter 1

Introduction

Software effort estimation is the task of approximating the amount of work required to develop a software project. This estimate is made using a unit of effort such as work-months, the amount of work accomplished by one employee working full time for a month. Effort estimates can then be converted into cost estimates by factoring in the salaries of the employees. The primary reason estimates are first made in terms of effort instead of cost is that it makes the estimates and data involved more compatible across time periods, geographic locations, and domains of the industry. For example, project data and effort estimates from 10 years ago can be useful without having to adjust for inflation.

Software effort estimation is not a precise science. Mistakes are common and those mistakes can waste hundreds of millions of dollars. For example:

> *"To gain control over its finances, NASA last week scuttled a new checkout and launch control system (CLCS) for the space shuttle. A recent assessment of the CLCS, which the space agency originally estimated would cost $206 million to field, estimated that costs would swell to between $488 million and $533 million by the time the project was completed.*
> *" – June 11 2003, Computer News*

The CLCS experience is hardly unique. Boehm cautions that early life-cycle estimates of software effort can be inaccurate by up $\pm$400% [8, p310]. Molokken and Jorgensen report that, according to their survey, 60-80% of the time a software project will overrun its estimate by an average of 30% [98]. The 2001 Standish Group ”Chaos Report” [127] showed an average overrun of 45%. In addition, they [127] found only 28% of projects to finish on time and on budget. Unfortunately, many of these successful projects were far under budget [127]. A study [101] by Ostvold et al. of the Norwegian software industry found 76% of projects had effort overruns with a mean overrun of 41% and a median overrun of 21%. They also found that estimates for internally developed projects were much more accurate than externally developed projects [101]. Other authors have cautioned that off-the-shelf “un tuned” effort estimation models have been up to 600% inaccurate in their estimates, e.g. [102, p165] and [58]. Clearly there is a strong need for better software effort estimation methods.

There are many software effort estimation approaches, which are outlined in sections 2.1, 2.2, and 2.3. This thesis focuses on the use of the COCOMO model and variations of this model due to the public description of the algorithm, available data, as well as prior use and research by the research client, NASA’s Jet Propulsion Laboratory. COCOMO is a popular model, developed by Barry Boehm [8,10] of USC, which defines a linear relationship between effort and code size. The model includes several cost drivers, such as programmer capability and required reliability, which affect the estimate either as a linear scalar value, or as a slight exponential change. COCOMO is typically calibrated on the historical data local to an organization, but it has predefined calibration values which may be used in the absence of data.

Although there are several approaches to software effort estimation, the industry uses mostly expert judgment to make their estimates [44]. One of the major goals of this thesis is to make progress towards a conceptual framework of integration of expert and model-based methods. There are several ways this can be done. One way is to explicitly develop a method that combines experts and models. The simplest way to do this would be to combine estimates from each, or develop a radically new approach of integration from scratch. Another way would be to either look at model-based methods and add aspects of expert-based methods, or vice versa. This would allow some benefits of integration without having to build a new model from scratch. In this thesis, the second approach which adds aspects of expert-judgment to a model paradigm was chosen because this research was in collaboration with the Jet Propulsion Laboratory which is required to provide model-based estimates.

What is a good way to adds aspects of expert-judgment to a model-based paradigm? Jorgensen [49] has described 12 expert-based best practices. These expert best practices could be automated in a model to add aspects of good expert judgment to the model. Jorgensen's 12 expert-based best practices are:

1. evaluate estimation accuracy, but avoid high evaluation pressure;
2. avoid conflicting estimation goals;
3. ask the estimators to justify and criticize their estimates;
4. avoid irrelevant and unreliable estimation information;
5. use documented data from previous development tasks;
6. find estimation experts with relevant domain background;
7. estimate top-down and bottom-up, independently of each other;
8. use estimation checklists;
9. combine estimates from different experts and estimation strategies;
10. assess the uncertainty of the estimate;
11. provide feedback on estimation accuracy; and,
12. provide estimation training opportunities.

Jorgensen strongly endorses expert predictions by claiming that formal models should not replace expert judgment, but should instead support expert prediction [53]. The justification for this is that, ”expert predictions may include detailed knowledge about the maintainer and the environment which it is not practical to include in a formal model [53].” However, Jorgenson is silent on how this might be best accomplished. The challenge of this thesis was to see how to integrate automated model-based support with the above list.

Total coverage of the whole list is not possible. For example, the second item, *avoiding conflicting estimation goals*, remains in the domain of the user, and it is up to them to properly use the model. However, automated model-based support tools can be offered for most of this list.

How is it that these expert practices might be included in a model? The first practice may be accomplished by evaluating the model with historical data. The third practice is accomplished by basing estimates off of past experience via historical data, as well as by using validating the method with historical data, and assessing the uncertainty in the estimate. The fourth practice may be accomplished by using feature and record selection algorithms to prune irrelevant and unreliable information. The fifth practice is inherently accomplished by any model calibrated on historical data. The sixth practice, "find estimation experts with relevant domain background", can be thought of as using the best models available with relevant calibration data. The seventh practice would require a lot of development to create the bottom-up assessment tools, but it could be done. The eighth practice could be implemented by requiring specific steps be taken in the application. The ninth practice could be done by stacking, or combining, the estimates of different models. The tenth practice may be handled by representing each source of uncertainty and providing a range or distribution of estimates instead of a single number. The eleventh practice may be accomplished by making an estimate for a project in which you don't know the actual effort, and then having the actual effort revealed to you. Finally, providing estimation training opportunities could be built into a model as an interactive training guide.

In addition, involving the cost analyst, or expert, in the model's operation further increases the integration of the hybrid approach. This thesis contends that the model's operation shouldn't be viewed as a "black box" from the expert, allowing them to be ignorant of the model's decision making structure. Rather, if the expert understands both how the model works and can see and guide its operation, then much greater insight into the final estimate is obtained.

What model based methods should be included in the hybrid combination? As mentioned before, this thesis focuses on the COCOMO family of regression models due to available experimental datasets. Other models may well be worthy of inclusion in a hybrid system, but were not investigated in this thesis. This thesis investigates several methods to improve the COCOMO model which may be worth using including feature subset selection, bagging, and boosting. Feature selection tries pruning unnecessary, irrelevant, and noisy features to improve performance and create a simpler model. It is described in section 2.6 and experiments using it are found in section 3.2. One benefit of feature selection is that it helps satisfy Jorgensen's fourth best practice. Bagging and boosting, described in sections 2.8.2 and 2.8.3, build several deviations of the same model and combine their estimates to achieve a theoretically better estimate. Boosting does this by focusing on the training instances that are the most difficult to classify. Our prior research [95] found a small number of severe outliers in the estimation errors. Thus, this technique is appealing for cost estimation. Before using these techniques, their efficacy should be demonstrated with existing effort estimation data. Thus, the Chapter 3 details research and experiments on feature subset selection, bagging, and boosting of the COCOMO model. In addition, Jalali [46] has found an automated record selection algorithm named LOCOMO to be useful for COCOMO estimation, which helps satisfy practice # 4. For this reason as well as its performance gains, LOCOMO is included in model based methods used in the hybrid combination.

Another important aspect of this thesis is the representation of uncertainty in the estimate. Most software effort estimates are made using a point estimate, however software effort estimation is widely recognized as an inherently uncertain task [37,52,54,68,72,103,109,118]. Thus, project managers should be provided with a representation of the uncertainty in the estimate as well so that they may consider the risk of an overrun when planning a project. Kitchenham and Linkman have described four sources of estimate uncertainty [68].

- *Measurement error* is error from accuracy limitations in the input variables of the model.
- *Model error* is error due to the model's abstractions from reality.
- *Assumption error* is error from making incorrect assumptions about a model's input parameters.
- *Scope error* is error from estimating outside the model's domain.

To the best of our knowledge, no estimation strategy incorporates all four of Kitchenham's sources of uncertainty. This thesis provides a more comprehensive handling of these uncertainty operators and offers methods that may reduce the actual uncertainty.

Molokken-Ostvold et al. [101] found that most research treats an estimate as a single point, but in industry there are several estimates. This is because estimates are needed at different stages of development. For example, an early estimate with ambiguous requirements may be made, but later a requirements specification is produced and a new estimate is made.

Considering the need for estimates at different stages in the software life cycle, an estimation model would benefit from being usable and effective at each stage. Thus, one of the goals of this thesis is to provide estimation methods useful at different software life-cycle stages.

Grimstad et al. [36] recommend the use of a consistent effort estimate terminology among researchers. Molokken-Ostvold et al. suggest, "If the goal is to investigate the estimation accuracy of professionals in a company", then, "it is meaningful to use the most likely estimates at the planning stage, instead of, for example, early estimates communicated to clients." [101] A most-likely estimate has equal likelihood of being above or below the actual estimate. This contrasts with other types of estimates such as estimates to win a bid or to budget a project. For example, when budgeting a project, a value above the most-likely, or 50 percentile estimate, is typically used to increase the likelihood of finishing the project. In this thesis, the accuracy of the estimate is important when comparing methods. The most-likely estimate will be closer to the actual estimate more often than an adjusted budget estimate. Thus, the estimates made in this thesis refer to most-likely estimates.

The results of this thesis research have been implemented in a tool known as 2CEE for NASA's Jet Propulsion Laboratory. An important quality of this thesis is the application, feedback, and validation of the novel research methods in an industrial environment. Why is this important? In a recent review of software effort estimation studies, Jorgensen and Shepperd discovered a disconnect between research and the actual use of effort estimation methods [52]. By receiving industrial feedback on the research of this thesis, and by validating the methods in the industrial environment, this thesis makes a step toward breaching the research to reality divide.

The 2CEE tool increases the experts involvement with the model, and it implements 7 of Jorgensen's 12 expert based best practices. This makes a good improvement towards a hybrid approach to expert and model based effort estimation. In addition, it represents the uncertainty of the estimate, and provides numerous calibration options. The tool is currently available from the Jet Propulsion Laboratory. To obtain a copy, send an email to `softwarerelease@jpl.nasa.gov` with your name, citizenship, affiliation, and the name of the software that you are requesting, 2CEE.

1.1 Contribution of This Thesis

This thesis contributes a variety of finds to the literature, including:

- A new approach to estimation that increases expert involvement with the estimation model.
- Integration of 7 of Jorgensen's 12 expert judgment best practices into an estimation model.
- A more comprehensive representation of estimate uncertainty by considering the sources of uncertainty.
- A near linear time feature subset selector providing improvements just as good as an exhaustive search.
- An industrial case study of the development and introduction of these methods at NASA's Jet Propulsion Laboratory.
- Much different COCOMO calibration coefficients than have previously been observed.
- One of the few studies to evaluate software effort estimation methods using nonparametric analysis due to the non-Gaussian distribution of errors.

1.2 Structure of This Document

The remainder of this document is organized as follows:

- Chapter 2 is a literature review describing related work
- Chapter 3 details some laboratory studies using some techniques described in the related work
- Chapter 4 describes a new software effort estimation methodology called *2CEE*
- Chapter 5 details the implementation and experimental validation of 2CEE in the industry
- Chapter 6 concludes the study by summarizing the accomplishments towards each thesis goal, and describes future work

In addition, an appendix containing more experimental results is freely available online at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`.

Chapter 2

Related Work

2.1 Software Effort Estimation Taxonomy

There are many types of software effort estimation models. Myrtveit et al. have described a hierarchy [106] of these models as shown in figure 2.1. Myrtveit's taxonomy groups estimation models into *sparse-data methods*, which need little to no historical data, and *many-data methods*, which need significantly more data records [106].

2.2 Sparse Data Methods for Software Effort Estimation

Myrtveit et al. list three types of sparse-data methods for software effort estimation: [106]

- *Expert Judgment*
- *AHP* - Analytic Hierarchy Process
- *CBR* - Case Based Reasoning

2.2.1 Expert Judgment

In software effort estimation, expert judgment refers to the creation of an estimate where, "a significant part of the estimation process is based on a non-explicit and non-recoverable reasoning

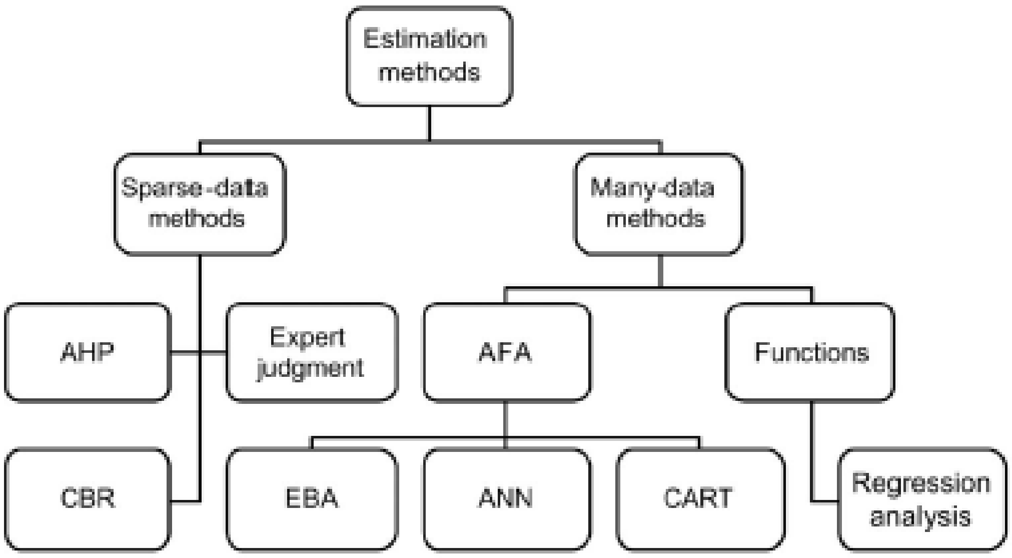


Figure 2.1: Myrtveit's taxonomy of software effort estimation methods. [106]

process, i.e., intuition [49]." Its use ranges from, "unaided intuition", to, "expert judgment supported by historical data, process guidelines and checklists. [49]"

Roughly 21% of research in software effort estimation between 2000 and 2004 involved the expert judgment estimation approach [52]. Although researched less than some other methods, expert judgment is highly prevalent in the industry. In 1991, Hihn and Habib-Agahi's reported that only 7% of software effort estimators primarily used formal models [44]. Conversely, 83% primarily used informal analogy, 4% formal analogy, and 6% rules of thumb [44]. Here, informal analogy was expert judgment where, "documented data was not used to support the estimate", formal analogy was expert judgment that considered documented data, and rules of thumb may have been, "derived from actual project data", or from expert opinion [44].

Similarly, other reports indicate that although expert judgment research is low, its use in industry ranges from 62%, 72%, 84%, to 86% prevalent [49]. It is possible that expert judgment's dominance in the industry is because it leads to better estimates. Jorgensen considered this and investigated fifteen studies using both expert and model-based methods [49]. He found that of, "the fifteen studies, we categorize five to be in favour of expert estimation, five to find no difference, and five to be in favour of model-based estimation [49]." Perhaps this conclusion instability is due to the same problems discussed in section 2.5.3.

Considering the popularity of expert based approaches, it is important for the potential effort estimator to know how to properly utilize this method. Jorgensen offers twelve ”best practice” guidelines for expert based software effort estimation [49, 50]. For more on these best practices, see section 2.4.

Expert-based estimation approaches often utilize multiple experts to improve the estimate. Passing and Shepperd [108] discuss the group approach:

> ”There are two main reasons why group discussions are supposed to improve expert estimation. First, group discussions lead to the acquisition of new knowledge through group discussion. And second, exchanging assumptions and views on possible threats to the estimates can lead to better estimates, because more aspects are taken into account than group members could have determined individually [60, 77, 99].
>
> There are various group techniques available. The one most often mentioned for estimation meetings is the Wideband Delphi Technique [8]. It is an iterative process aimed at avoiding group pressure effects through anonymous estimates. Group members discuss relevant issues openly, but do not have to agree on a consensus estimate. Instead, each member notes his personal estimate after the discussion. This has been found to reduce possible compliance to a majority opinion [4]. The personal, anonymous estimates are then given as an input to the following round. The goal is to seize the positive effects of group discussions as mentioned above, while avoiding compliance to group pressure or dominant individuals [77].” [108]

Passing and Shepperd [108] found improved accuracy using a group of experts instead of one. Another benefit reported by Molokken and Jorgensen is that groups of experts produce less optimistic and more realistic estimates than individual expert estimates [100]. Considering that most estimates are overly optimistic, this is a useful benefit. For more on this phenomenon, see section 2.5.6.

Jorgensen strongly endorses expert predictions by claiming that formal models should not replace expert judgment, but should instead support expert prediction [53]. The justification for this is that, ”expert predictions may include detailed knowledge about the maintainer and the environment which it is not practical to include in a formal model [53].”

Combining Expert Judgment and Model-Based Estimation

Many effort estimators staunchly support one ideology: for example, either expert judgment or model-based estimates. Some conduct more than one estimate, using one to support the other. Others combine estimates from multiple approaches. A few integrate the multiple methods into one.

Meli [86] has suggested such an integrated approach combining expert judgment with models:

> ”People and methods are tremendously important to gain a high quality in the estimation results. Instead of conceiving them as alternative means we should consider them as synergic resources. This may happen if we are supported by a conceptual framework of integration and a set of operational rules to follow.” [86]

One of the goals of this thesis is to make progress towards the, ”conceptual framework of integration”, suggested by Meli. A recent review [118] by Shepperd echoes Meli’s sentiment:

> ”Another area in which more research is needed is the less formal combining of prediction systems with expert judgement. In the past the implication if unvoiced including from some of the authors work has been that formal prediction systems will one day replace experts. However this is extremely unlikely not least because software project cost predictions are infrequent but very high value decisions. Hence we need more work to consider how formal models might support and assist experts rather than replace them.” [118]

One method with characteristics of both expert and model based systems is the *Analytic Hierarchy Process*.

2.2.2 Analytic Hierarchy Process

The *Analytic Hierarchy Process* [116], *AHP*, is a decision-making technique used in many disciplines for, ”dealing with problems which involve the consideration of multiple criteria simultaneously. [112]” Although it’s use for software effort estimation technically qualifies AHP as an expert judgment approach, As a technique for software effort estimation, AHP combines historical data and expert opinion by quantifying subjective judgment. Shepperd and Cartwright have found

AHP to provide good software effort estimates, possibly because it allows the expert to, ”view the problem in a more structured and systematic way. [121]”

The AHP involves building a hierarchy of the problem with the goal as the root and the relevant attributes are placed in the level(s) below it [112]. Weights are assigned to each node and many pairwise comparisons and matrix multiplications are made [112]. It is beyond the scope of this study to describe the details of calculating these weights. Next, The end result is a method which, ”provide[s] a formal, systematic means of extracting, combining, and capturing expert judgments and their relationship to analogous reference data [43, 116].” An evolution of this technique for software effort estimation is the use of *probabilistic* pairwise comparison matrices as suggested by Hihn and Lum [43] to include uncertainty in the expert-based estimates. It is important to assess the uncertainty of an estimate to understand the risk involved; providing a range of estimates instead of a point estimate is an intuitive way to do this. The problem of uncertainty is discussed in more detail in section 2.5.4.

2.2.3 Case Based Reasoning

Case-based reasoning, CBR, is machine learning technique that may be used for effort estimation. Kolodner define *case-based reasoning*:

> ”Case-based reasoning systems are systems that store information about situations in their memory. As new problems arise, similar situations are searched out to help solve these problems. Problems are understood and inferences are made by finding the closest cases in memory, comparing and contrasting the problem with those cases, making inferences based on those comparisons, and asking questions when inferences can’t be made.” [71]

CBR is well received among researchers because it, ”has an intuitive appeal for use in software effort estimation as it has the capability to model the way expert estimation is performed as well as explaining the reasoning applied to adapt past cases.” [30] CBR follows a cycle of four basic processes [2]:

1. RETRIEVE the most similar case or cases

2. REUSE the information and knowledge in that case to solve the problem

3. REVISE the proposed solution

4. RETAIN the parts of this experience likely to be useful for future problem solving

The CBR approach doesn't require constraints on the information recorded about software projects. This contrasts with models like COCOMO [8, 10] that predefine what information to collect about software projects. CBR, "allow[s] estimators the freedom to utilise those features that they believe best characterize their projects and are most appropriate to their environments [57]." It is no surprise then, that feature subset selection methods are very useful for case-based reasoning models [57, 63, 64]. Delany et al. [24] write that there, "is a need to identify abstract features that capture similarity across domains."

At its lowest level, CBR often boils down to a nearest neighbor algorithm. For example, a CBR model may use the, "*k*-nearest-neighbor strategy, where some fixed, small, number *k* of nearest neighbors–say five–are located and used together to [estimate] [135]." This introduces some algorithmic choices to make. For example, what is the best value of *k*? Determining the nearest neighbors usually involves calculating an *n*-dimensional distance calculation between the new project and each past project, where *n* is the number of features. Normalization is commonly done equalize the impact of each feature on the distance measure.

CBR has been researched for software effort estimation for many years [24, 30, 57, 63, 64]. For example, Shepperd et al. developed *ANGEL* [122], which is a CBR tool for software effort estimation that works with, "arbitrary sets and types of features." Thus, ANGEL does not make an assumption about the form of the relationship between the data and the estimate. ANGEL can be thought of as a nearest neighbor algorithm that estimates a new project by reasoning from similar cases. It normalizes features such that, "the influence of a feature is not related to the choice of unit [64]." Shepperd and Schofield, "found that ANGEL performed as well or better than a stepwise regression model across 9 datasets [64, 120]."

CBR sounds a lot like EBA. Indeed, Aamodt and Plaza equate one to the other, but with one slight difference:

> "Analogy-based reasoning: This term is sometimes used, as a synonym to case-based reasoning, to describe the typical case-based approach. However, it is also often used

to characterize methods that solve new problems based on past cases from a different domain, while typical case-based methods focus on indexing and matching strategies for single-domain cases." [2]

This explains why Myrtveit et al. grouped EBA as a many-data approach and CBR as a sparse data approach [106]: CBR typically considers similar in-house data, while EBA considers much more by casting its net into cross-company data. Unfortunately, this distinction is not required or clear to many researchers, so CBR is essentially the same as EBA.

2.3 Many Data Methods for Software Effort Estimation

Myrtveit et al. describe two types of many-data estimation methods: *AFA's* and *Functions* [106]. In this context, *Functions*, assume, "there is a mathematical relationship between the variables expressed in a formula", typically of, "the general form $y = Ax^B$ [106]." *AFA's*, or *Arbitrary Function Approximators*, do not make this assumption.

2.3.1 Arbitrary Function Approximators

Myrtveit et al. list three types of AFA's for software effort estimation: [106]

- *EBA* - Estimation by Analogy
- *ANN* - Artificial Neural Networks
- *CART* - Classification And Regression Tree

Estimation by Analogy

Estimation by Analogy is similar to case-based reasoning, although it has a connotation that implies the use of more relevant data from other organizations [2]. For more on this, see section 2.2.3.

EBA is similar in principle to the other methods described except it does not make a functional assumption about the relationship between project data and effort. It can be thought of as a nearest neighbor approach that makes an estimate using the actual efforts of the historical records deemed relevant. This is done by describing the novel project being estimated, and then using this description to find similar historical projects. The actual efforts from these past projects are, ”then used, possibly with adjustment, to generate the predicted value. [122]” *EBA* is a popular technique which is heavily research for software effort estimation [76, 82, 105, 119, 126, 128]. A study comparing estimation by analogy with regression based estimation was unable to consistently determine one method superior to the other [82].

Artificial Neural Networks

> ”A *neuron* is a cell in the brain whose principal function is the collection, processing, and dissemination of electrical signals. The brain’s information-processing capacity is thought to emerge primarily from *networks* of such neurons.” [114]

Artificial Neural Networks, or ANNs, are machine learning algorithms that mimic the human brain in order to solve problems in a myriad of domains. They, ”provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples [97].” One potential use for an ANN is software effort estimation. Using ANNs to estimate software effort is not a new idea. Jorgensen’s *Systematic Review of Software Development Cost Estimation Studies* [52] found 11 studies using ANN for software effort estimation from 1990-1999, and another 11 studies from 2000-2004. In 1995, Srinivasan and Fisher [125] found ANN to be, ”competitive with SLIM, COCOMO, and FUNCTION POINTS as represented in a previous study [58] by Kemerer.” A simulation study by Finnie and Wittig [30] showed good software effort estimates by ANN. Conversely, a more recent simulation study by Shepperd and Kadoda found ANN had the least number of wins compared to stepwise regression, rule induction, and case-based reasoning [119]. Even more recently, Tronto et al. found ANN to outperform COCOMO and some other popular models [23].

Classification And Regression Tree

The *Classification and Regression Tree*, originally by Breiman et al. [12], is a nonparametric decision tree machine learning technique. In the context of software effort estimation, the goal is not to classify between different distinct classes, but is instead to estimate the effort. Thus the relevant aspect of CART in this context is the regression tree, which has a numeric value at each leaf node.

CART works by first building a decision tree until a ”maximal” tree has been made that overfits the training data [75]. Then, the tree is simplified by pruning unnecessary branches [75]. Finally, a tree is created which ”fits the information in the learning dataset, but does not overfit the information, is selected from among the sequence of pruned trees [75].”

CART, or more specifically regression trees have seen limited use for software effort estimation [14–17, 67, 125] In 1995, Srinivasan and Fisher [125] found CART to be, ”competitive with SLIM, COCOMO, and FUNCTION POINTS as represented in a previous study [58] by Kemerer.” In 1998, Briand et al. found [17] that, ”despite a general poorer accuracy [than a least-squares regression model], the regression tree models are more intuitive and easy to use for benchmarking purposes.” However, a year later, Briand et al. found [16], ”simple CART models perform a little better than other modeling approaches.”

2.3.2 Function Based Software Effort Estimation

The only type of function based software effort estimation shown by Myrtveit et al.’s taxonomy (figure 2.1) is regression analysis [106]. *Regression* is the act of, ”learning a continuous function [114].” Regression is the most heavily researched form of software effort estimation [52]. Roughly 51% of research in software effort estimation between 2000 and 2004 involved a regression based estimation approach [52]. Regression was found [47] to have the lowest median MRE of several methods using, ”a large-scale industrial data set which is professionally maintained by the International Software Standards Benchmarking Group (ISBSG).” There are many regression based effort estimation models [3, 8, 10, 21, 52]. This thesis focuses on the COCOMO family of regression models due to available experimental datasets.

COCOMO

The experimental case study data for this thesis uses COCOMO-format data. COCOMO, the Constructive Cost Model, was originally developed by Barry Boehm in 1981 [8] and was extensively revised in 2000 [10]. 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 is that unlike many other costing models such as SLIM or SEER, COCOMO is an open model with numerous published data [8, 10]. COCOMO measures effort in calendar months where one month is 152 hours which includes development and management hours. The core intuition behind COCOMO-based estimation is that as a program grows in size, the development effort grows exponentially. More specifically: In addition, the COCOMO II model, "has been validated using JPL data and provides reasonable flight and ground software estimates in the JPL environment [78]."

$$effort(personmonths) = a * \left(KLOC^{b}\right) * \left(\prod_{j} EM_{j}\right) \tag{2.1}$$

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.) [10, 48]. 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 2.1, EM_j is one of the many *effort multipliers* such as *cplx* (complexity) or *pcap* (programmer capability). Figure 2.1 describes the COCOMO 81 effort multipliers and illustrates which positively and negatively influence the effort estimate. Increasing the *upper* and *lower* groups of variables will *decrease* or *increase* the effort estimate, respectively. 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 (best practice #13 in figure 2.6). Table 2.2 shows Boehm's *EM* values for COCOMO 81.

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 2.2). COCOMO-II ignores these distinctions. This study used data sets in both the COCOMO 81 and COCOMO-II format.

COCOMO II is Barry Boehm's 2000 improvement to COCOMO [10]. In COCOMO II, the exponential COCOMO 81 term *b* was expanded into the following expression:

$$b + 0.01 * \sum_{j} SF_{j} \tag{2.2}$$

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

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

Table 2.1: COCOMO Features from [8]. Most range from “vl” for “very low” to “xl” for “extremely high”.

		vl	l	n	h	vh	xh
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

Table 2.2: The COCOMO-I β_i table [8]. For example, the bottom right cell is saying that if CPLX=xh, then the nominal effort is multiplied by 1.65.

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.

Figure 2.2: Standard COCOMO 81 development modes.

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 2.2 as well as some modifications to the list of effort multipliers, as shown in figure 2.3, and their associated numeric constants, as shown in table 2.4. Despite these improvements, COCOMO 81 is still used for research because of its publically available data. For more on this, see section 3.1.1.

Learning with Linear Regression

Linear regression assumes that the data can be approximated by one linear model that includes lines of code (KLOC) and other features f seen in a software development project:

$$effort = \beta_0 + \sum_i \beta_i \cdot f_i$$

Linear regression adjusts β_i to minimize the *prediction error* (the difference between predicted and actual values for the project).

Boehm argues that effort is exponential on the amount of code KLOC [8]:

$$effort = a \cdot KLOC^b \cdot \prod_i \beta_i$$

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

Figure 2.3: 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	

Figure 2.4: The precise COCOMO II numerics.

(where a and b are domain-specific constants). Such exponential functions can be learned via linear regression after they are converted to the following linear form:

$$log(effort) = log(a) + b \cdot log(KLOC) + \sum_i log(\beta_i)$$

Local Calibration

Local calibration (LC) can refer to any local tuning of a regression model. However, in this context, it refers to a specialized form of the linear regression of COCOMO developed by Boehm [8, p526-529].

Recall that COCOMO assumes project effort is exponential on KLOC; i.e.

$$effort = a \cdot KLOC^b \cdot \prod_i \beta_i$$

Figure 2.2 shows the β_i values recommended by Boehm (the names on the left hand side are defined in figure 2.1).

To operate, LC linearizes the exponential equation to generate

$$log(effort) = log(a) + b \cdot log(KLOC) + \sum_i log(\beta_i)$$

Linear regression would try to adjust all the β_i values. This is not practical when training on a very small number of projects. Hence, LC fixes the β_i values while adjusting the $< a, b >$ values to minimize the prediction error. We shall refer to LC as "standard practice" since, in the COCOMO community at least, it is the preferred method for calibrating standard COCOMO data [10].

SCAT

SCAT, short for Software Cost Analysis Tool, is a state of the art effort estimation tool developed and used by the NASA Jet Propulsion Laboratory [79, 80]. It has been used as JPL's standard software effort estimation method since 2003.

SCAT is part of a class of probabilistic software estimation methods that aim to represent the uncertainty of the estimate. This class of methods includes the use of fuzzy logic [84, 103, 131], Bayes algorithm [21, 109], and Monte Carlo simulation [5, 13]. More specifically, SCAT is part of the latter group of methods that use Monte Carlo simulation. Most of the probabilistic software

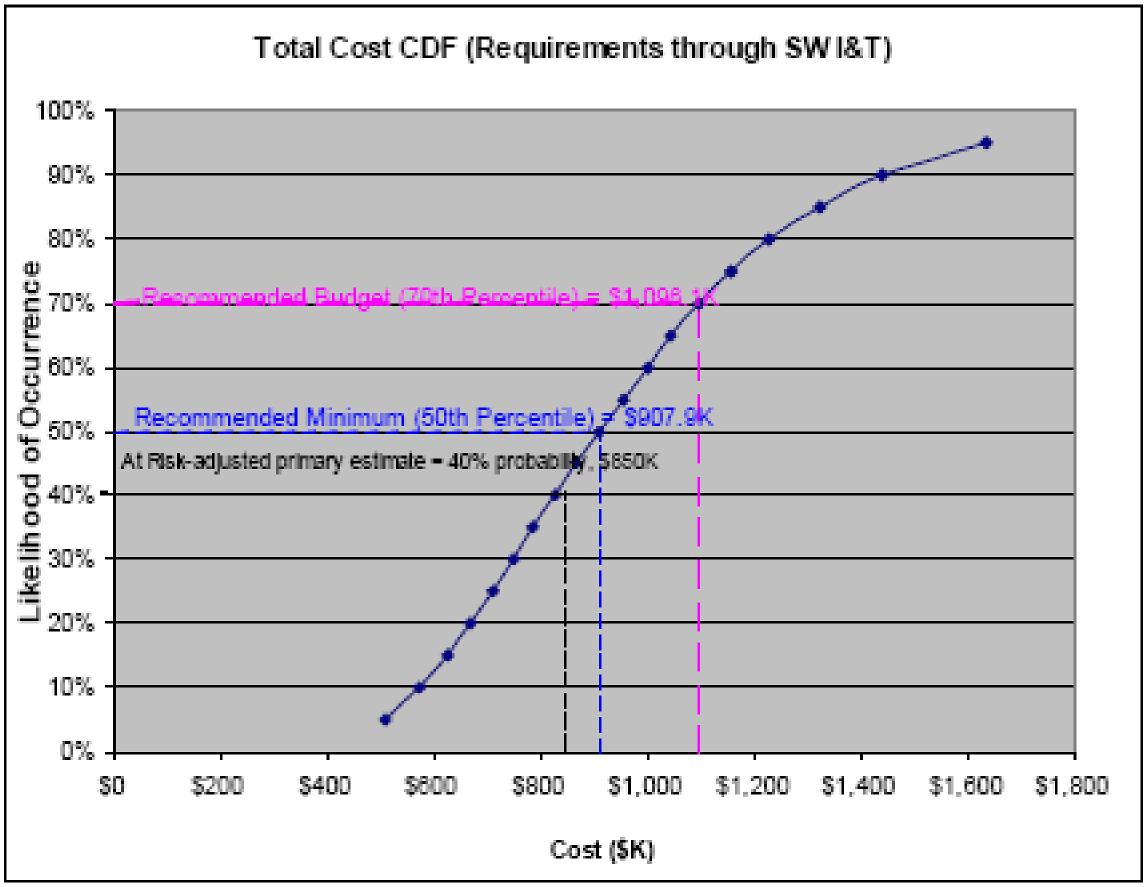


Figure 2.5: Example of SCAT's probabilistic output. [80]

estimation methods internally represent the model's variables with some form of distribution or range. On the other hand, since Monte Carlo simulation samples discrete instances from a distribution and calls an existing model that uses discrete input, it is a natural choice for existing models such as COCOMO that use discrete inputs. This is the technique chosen by SCAT, being, "an adaptation to COCOMO II that allows the use of a range of inputs (Low, Most Likely, High) to capture uncertainty and generate probability distributions through the use of Monte Carlo techniques [79]." SCAT does not work directly with historical calibration data but is instead set with a fixed linear and exponential COCOMO coefficient. The topic of uncertainty is described in more detail in section 2.5.4.

An example of the probabilistic output of SCAT is shown in figure 2.5. Note that JPL advises use of the 70% confidence estimate. This increases the likelihood of offering a sufficient budget.

2.4 Software Effort Estimation Best Practices

Software contracts often stipulate the use of industry *best practice*. In addition, most working professionals will want to use the best practices available even if not legally required. What is included in the best practice isn't always easy to define. In addition, once a ”best practice” is found and chosen to be implemented, there is usually resistance to the process change within an organization. Keung et al. [61] recognize this problem and suggest that, ”managers should develop a set of strategies to incrementally change the process according to the needs of the organisation.”

Laird offers three golden rules of estimation [72]:

- Require all estimates to be justified. Gut feeling is not an adequate justification.
- Dont use methods or tools blindly. Try estimating previous (completed) projects to validate and tune the methods.
- Educate your estimators. Knowing how to do something doesnt mean you know how long it will take. Train people in estimation. Accuracy is correlated with training and the ability to see results, not development experience.

An expanded list of software effort estimation best practices compiled by Menzies et al. [93] is shown in figure 2.6. One of the goals of this thesis is to utilize as many of the best practices as is practical.

2.5 Open Issues in Software Effort Estimation

2.5.1 Inaccuracy

”There are really only two kinds of estimates: “Lucky” or “Lousy.”” - Standish Chaos Report 2001 [127]

Section 1 clearly describes how notoriously inaccurate software effort estimates can be. What is the source of all of this inaccuracy? Jorgensen and Molokken-Ostvold found the source of the error changes depending on who you ask: the sources of error were typically factors outside of the control of those asked, and the sources of an accurate estimate were typically factors inside of the control of those asked [55].

According to Jorgensen [49], expert-based best practices include:

1. evaluate estimation accuracy, but avoid high evaluation pressure;
2. avoid conflicting estimation goals;
3. ask the estimators to justify and criticize their estimates;
4. avoid irrelevant and unreliable estimation information;
5. use documented data from previous development tasks;
6. find estimation experts with relevant domain background;
7. estimate top-down and bottom-up, independently of each other;
8. use estimation checklists;
9. combine estimates from different experts and estimation strategies;
10. assess the uncertainty of the estimate;
11. provide feedback on estimation accuracy; and,
12. provide estimation training opportunities.

According to Boehm [8, 10]; Chulani [21, 115]; Kemerer [58]; Stutzke [130]; Shepperd [120]; our own work [19, 20, 92]; and a recent tutorial at the 2006 International Conference of the International Society of Parametric Analysts [1], best practices for model-based estimation include at least the following:

13. *Reuse regression parameters* learned from prior projects on new projects;
14. *Log-transforms* on costing data before performing *linear regression* to learn log-linear effort models;
15. *Model-tree* learning to generate models for non-linear relationships;
16. *Stratification*, i.e. given a database of past projects, and a current project to be estimated, just learn models from those records from similar projects;
17. *Local calibration*, i.e. tune a general model to the local data via a small number of special tuning parameters;
18. *Hold-out* experiments for testing the learned effort model [92];
19. Assessing effort model uncertainty via the *performance deviations* seen during the hold-out experiments of item #17;
20. *Variable subset selection* methods for minimizing the size of the learned effort model [19,20,63,96].

This separation of model-based and expert-based methods is not a strict division since some practices fall into both categories: e.g. #4 & #20 are similar as are #10 & #19. Also, one way to view model-based methods is that they seek algorithms to make maximal use of #5. Further, some research actively tries to combine the two approaches:

21. Shepperd's *case-based reasoning* tools [120] explore algorithmic methods for emulating expert analogical reasoning;
22. Chulani & Boehm's *Bayesian tuning method* [21] for regression models allows an algorithm to carefully combine expert judgment with the available data;
23. This paper will argue for the use of heuristic *rejection rules* to represent expert intuitions on how to rank different effort models.

Figure 2.6: Three different categories of effort estimation best practices: expert-based (at top); model-based (in the middle); methods that combine expert and model-based (at bottom).

Ideally estimation models should be made to be as accurate as possible. One of the goals of this thesis is to improve estimation accuracy. Unfortunately, this is an inherently difficult problem, but why should it be? To answer this question, some researchers [37, 72] refer to a quote from Alfred Pietrasanta of the IBM Systems Research Institute in 1968:

> ”Anyone who expects a quick and easy solution to the multi-faceted problem of resource estimating is going to be disappointed. The reason is clear: computer program system development is a complex process; the process itself is poorly understood by its practitioners; the phases and functions which comprise the process are influenced by dozens of ill-defined variables; most of the activities within the process are still primarily human rather than mechanical, and therefore prone to all the subjective factors which affect human performance.” [110]

Nevertheless, it is inspiring that the existing models predict as well as they do. For example, figure 5.2 on page 94 shows the methods described later in this thesis produced a median MRE of 5.4% when estimating the software development effort of flight based NASA projects.

2.5.2 Scarce Data

Menzies et al. [93] warn that there is not enough data available to properly calibrate software effort estimation models. A common guideline for regression models is that 5 to 10 records are needed for each variable in the model [69]. Otherwise, the variations in the records have too strong of an impact an the regression results aren’t reliable. Thus COCOMO with its 16 variables should have at least 80 to 160 records. COCOMO II is even worse, needing at least 110 to 220 records to accurately calibrate to its 22 variables. Unfortunately, software effort estimation models typically do not have this much calibration data [10, 29, 92]. This is due to the slow generation of software projects and the desire to calibrate using similar, preferably in-house, data. This lack of data suggests the use of feature subset selection methods should provide more stable calibrations of effort estimation models by reducing the data necessary for regression.

It is possible to make software effort estimates without tuning to your organization’s local experience. Many estimation models may be used off the shelf without any local tuning. This approach doesn't guarantee the best performance, especially if your organization differs greatly from the industry as a whole, and may be as much as 600% inaccurate [58].

Another approach is to calibrate using similar projects from other organizations. Kitchenham et al. note that while many researchers have recommended developing models using only single-company data only, in some cases cross-company data will suffice [65]. However, they are still researching the conditions affecting the usefulness of cross vs within-company datasets. A study [134] by Wieczorek and Ruhe, "found that in general the local models developed using company-specific data do not perform significantly better than the global models." However, MacDonell and Shepperd recently concluded that more research was necessary to make a conclusion regarding the topic of cross vs within-company datasets [81].

Working with limited datasets brings with it some unique concerns. The good news is that some algorithms that would be too slow to run with larger datasets will run quickly with a small dataset. The bad news is that the scarce dataset makes it difficult to statistically justify algorithmic decisions, tuning the model, and even experimental results. The public effort estimation data available is also limited, making it difficult to generalize research claims from this data to the industry as a whole.

2.5.3 Method Evaluation

In order for new research to discover improved effort estimation methods, it is necessary to have an oracle that can evaluate the competing method's effectiveness. In our previous work, we caution that, "while the need for better estimates is clear, there exists a very large number of effort estimation methods [49, 52] and no good criteria for selecting between them [95]."

Parametric Evaluation

Parametric statistics assume the population fit some distribution, typically the normal distribution. The performance of models generating continuous output can be assessed in many ways using parametric evaluation, including PRED(30), MMRE, correlation, median MRE, 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. Given T tests, MMRE is calculated as follows:

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

The median MRE for a set of T tests is calculated by sorting the T tests and choosing the middle MRE value if T is odd, or the average of the middle two MRE values if T is even.

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. Given a test set of size T, correlation is calculated as follows:

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

$$corr = S_{pa}/\sqrt{S_p * S_a}$$

All these performance measures (correlation, MMRE, median MRE, and PRED) address subtly different issues [66]. Overall, PRED measures how *well* an effort model performs while MMRE measures how *poorly* it performs. 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 [120, p736].

Foss et al. report that MMRE is an, ”unreliable selection criteria”, that has a tendency to, ”prefer a model that underestimates [31].”

Nonparametric Evaluation

Nonparametric statistics do not make any assumptions about the distribution of the population. Most of these methods use some sort of rank-order technique.

There is significant cause to use nonparametric methods to evaluate software effort estimation methods. Mair and Shepperd were unable to find a consistent result when comparing regression based software effort models with analogy based approaches [82]. Most of the studies in their review used parametric statistics such as MMRE. Demsar [25] has recently warned that non-Gaussian populations are common enough to require a methodological change in data mining. In addition, Myrtveit et al. found that the best effort estimation model changed based on which parametric accuracy measure was used, and even how the cross validation was sampled [106]. Similarly, Menzies et al. noticed alarmingly large standard deviations of the MRE values in their software effort estimation experiments [90]. Their initial research into this phenomenon discovered that data pruning techniques [20, 45, 90] and some data mining techniques [93] reduce this deviation. Finally, our previous work reported a simple discovery that explains all of these inconsistent results: the effort estimation errors do not fit a Gaussian distribution, but rather have a small number of very large outliers [95]. Thus, we suggest using nonparametric statistical measures. After reviewing available nonparametric methods, we suggested the Mann Whitney Wilcoxon rank-sum test: [83]

Using this evaluation criteria, our previous work was, ”the first [report] to offer stable conclusions regarding effort estimation across such a wide range of methods [95].” One of the goals of this thesis is to evaluate methods using nonparametric evaluation. For an example of the use of the Mann-Whitney U test, see figure 2.7. For more on the use of nonparametric statistics in evaluating effort estimation models, see Jalali [46]. Our previous work suggests that, ”since we seek methods that can be rejected, the value of interest to us is how often methods *lose* [46, 95].”

The sum and median of A's ranks is

$$\begin{aligned} sum_A &= 1+2.5+5.5+7+9.5 = 25.5 \\ median_A &= 5.5 \end{aligned}$$

and the sum and median of B's ranks is

$$\begin{aligned} sum_B &= 2.5+4+5.5+8+9.5+11 = 40.5 \\ median_B &= 6.75 \end{aligned}$$

The U statistic is calculated from $U_x = sum_x - (N_1(N_2+1))/2$:

$$U_A = 25.5 - 5 * 6/2 = 10.5$$
$$U_B = 40.5 - 6 * 7/2 = 19.5$$

These can be converted to a Z-curve using:

$$\begin{aligned} \mu &= (N_1 N_2)/2 &&= 516.4 \\ \sigma &= \sqrt{\tfrac{N_1 N_2 (N_1+N_2+1)}{12}} &&= 5.477 \\ Z_A &= (U_A - \mu)/\sigma &&= -0.82 \\ Z_B &= (U_B - \mu)/\sigma &&= 0.82 \end{aligned}$$

(Note that Z_A and Z_B have the same absolute value. In all case, these will be the same, with opposite signs.)
If $abs(Z) < 1.96$ then the samples A and B have the same median rankings (at the 95% significance level). In this case, we add one to both $ties_A$ & $ties_B$. Otherwise, their median values can be compared, using some domain knowledge. In this work, *lower* values are better since we are comparing errors. Hence:

- If $median_A < median_B$ add 1 to both $wins_A$ & $losses_B$.
- Else if $median_A > median_B$ add 1 to both $losses_A$ & $wins_B$.
- Else, add 1 to both $ties_A$ & $ties_B$.

Figure 2.7: An example of the Mann-Whitney U test from Menzies et al [95].

Cross Validation

It is important to evaluate a method using different datasets for training, or model calibration, and testing the method's performance. Failing to do this risks making a claim about performance that will not generalize well to the actual performance in the real world. Unfortunately when there is not much available data, separating the dataset into a single training and test set is inadequate. In this case, a more advanced technique, *cross-validation*, should be used. A cross-validation experiment splits the data into a predefined number of equal partitions called folds [135]. Then each partition is used in turn for testing while the rest of the data is used for training. This process is done randomly several times in order to produce a stable performance score. The standard procedure for cross-validation is to use *10 by 10-fold cross-validation* and average the scores [135]. There is a, "debate [that] continues to rage in machine learning and data mining circles about what is the best scheme for evaluation [135]." Although 10-fold cross-validation is considered standard practice, other popular techniques include *leave-one-out* cross-validation and cross-validation through *bootstrapping*.

Leave-one-out cross-validation [135], also known as *n*-fold cross-validation, equates the number of folds to the number of instances in the dataset, *n*. *N*-fold cross-validation benefits from using as much data as possible in each training instance. This may be useful for especially small datasets because reducing the training data too much could handicap the learner. In addition, *n*-fold cross-validation is deterministic. Therefore, it is not necessary to repeat the cross-validation several times because there is no random sampling in the operation. This is useful for making easily repeatable experiments because there is no need to record the random number generator used and the seed used to initialize it. Unfortunately, *n*-fold cross-validation requires *n* calibrations and uses of the learning algorithm. When *n* is very high this computational cost can make this cross-validation technique impractical. Witten and Frank comment that, "leave-one-out seems to offer a chance of squeezing the maximum out of a small dataset and obtaining as accurate an estimate as possible [135]."

Bootstrapping is a popular statistical operation based on *sampling with replacement* that can be used for cross-validation. Sampling with replacement implies that when building a training dataset any record may be selected multiple times. To use this technique for cross-validation, many times a *bootstrap sample* is randomly generated. This bootstrap sample contains *n* records, where *n* is the number of records in the original dataset. However, each record in the bootstrap sample is randomly selected with replacement from the original dataset. Thus, the bootstrap sample will contain some of the original records multiple times, and others not at all. The testing dataset is simply all of the records not chosen in the bootstrap sample. In the unlikely occurrence that the original dataset is replicated as a bootstrap sample, there will be no testing set, so fail safes should be programmed to avoid this possibility. It has been shown mathematically that a typical bootstrap sample will contain 63.2% of the original data instances. Witten and Frank comment that, "the bootstrap procedure may be the best way of estimating error for very small datasets [135]."

2.5.4 Uncertainty

Musilek et al. have demonstrated that small errors in the COCOMO II model can lead to large changes in the estimate [104]. Clearly, to get an accurate effort estimate, relevant project metrics must be measured carefully and consistently.

> Size and effort multipliers are very important and adequate time and resources should be devoted to their accurate evaluation. The scale factors are much less important and could be neglected (set to their nominal values) if necessary.

[104]

The sensitivity issue shown by Musilek highlights a symptom of the larger problem of uncertainty. Indeed, software effort estimation is widely recognized as an inherently uncertain task [37, 52, 54, 68, 72, 103, 109, 118]. Assessing the uncertainty of the estimate is one of Jorgensen's twelve best practice guidelines for effort estimation [49]. This uncertainty may be a contributing factor to the deviance problem.

Beyond carefully collecting metrics, Musilek et al. proposed using fuzzy sets for the input variables, and concluded that the manager should, "no longer [be] left with a single number estimate that could be highly misleading [104]." This implies that the uncertainty of the input variables is gathered. This is an important task too often ignored by the effort estimation research community. Jorgensen has investigated the psychology of gathering metric ranges from developers, and found that merely changing the phrasing of the question changing the metric ranges [54].

> Instead of asking the estimators to provide the minimum and maximum effort values based on given confidence levels, e.g., ”almost sure” or ”90 percent confident,” it seems to be better to ask them to assess the probability of the actual effort being higher or lower than a certain value. For example, our results suggest that it is better to ask the estimator, ”How likely is it that the project will require more than 1.700 work-hours?”, rather than, ”What is the maximum cost of the project? Be 90 percent confident.” [54]

In addition, Jorgensen found that the developers might have other motives than realism when providing ranges of uncertainty. In the words of one of the developers in his study:

> I feel that if I estimate very wide effort minimum-maximum intervals, this indicates a total lack of competence and has no informative value for the project manager. I'd rather have fewer actual values inside the minimum-maximum interval, than providing meaningless, wide effort intervals. [54]

This is an unfortunate reality, especially considering that, ”low estimation accuracy is not necessarily an indicator of low estimation skills when the software development project work is highly uncertain [49, 56].”

Kitchenham and Linkman have described four sources of estimate uncertainty [68].

- *Measurement error* is error from accuracy limitations in the input variables of the model.
- *Model error* is error due to the model's abstractions from reality.
- *Assumption error* is error from making incorrect assumptions about a model's input parameters.
- *Scope error* is error from estimating outside the model's domain.

Another source of error mentioned by Doban and Pataricza [26] is a, ”statistically insufficient number or representativeness of the basic data set serving for extrapolation.” This is an important factor that is easily ignored. Tuned models should be given a sufficient amount of data for calibration. Boehm [10] recommends a minimum of 5 records for local calibration of his COCOMO model. Ideally there should be many more records.

This literature review has been unable to find any research actively addressing all of Kitchenham's uncertainty operators. However, many models have adapted to incorporate measurement error. This is done by representing the both the input variables and the output estimate as a range instead of point values. For example, Tian and Noore warn that the discretization of effort multipliers used in COCOMO into six categories results in an inherent uncertainty in the historical data, and thus methods such as fuzzy logic should be used to account for this inherent ambiguity [131]. This has been accomplished for effort estimation using fuzzy logic [84, 103, 131], Bayes algorithm [21, 109], and through monte carlo simulation [13]. Fuzzy logic and Bayes algorithm require developing an estimation model to work directly with their distributions, but the monte carlo simulation does not. This is because it simply samples discrete instances from a distribution and calls an existing model used to discrete input. Thus, monte carlo simulation is better suited for use with pre-existing models, while the other two methods should be considered if developing a new model.

A more severe source of measurement error is estimation with incomplete data. Although techniques exist for estimating effort with incomplete data [129], it is still advised to use complete datasets. In an impressive display of ability despite incomplete data, Menzies et al. considered uncertainty from a more positive angle and realized that if you can describe all of your model's input parameters using ranges found from industry, then it is possible to make estimates without calibration data at all [87, 88].

Further evidence of uncertainty in software effort estimation is the attribute instability problem identified by Menzies et al. [94] regarding Boehm's delta estimation method [9]. The delta estimation method proposed by Boehm works by basing the estimate off of similar previous projects and adjusting for the delta between the new and the old using the COCOMO cost drivers [9]. Each of Kitchenham's uncertainty operators can potentially explain this instability. The most unsettling potential cause is that of scope error in the case in which the true correlation between the attributes and target variable has indeed changed due to the different domain.

Jorgensen found experts tend to be overconfident about the uncertainty of their estimates [50]. A study by Gruschke and Jorgensen investigated the ability of experts to improve at uncertainty estimation using a feedback mechanic [37]. They found some experts could improve, however, they also found, ”that we cannot expect uncertainty assessments to improve when they are dominantly intuition-based [37].” This result suggests that more structured methods for uncertainty assessment should be used. Model-based effort estimation methodologies have the potential to excel at representing this uncertainty. One of the goals of this thesis is to represent estimation uncertainty by accounting for Kitchenham’s sources of estimation error. In addition, a goal of this thesis is to reduce estimation uncertainty.

User Distributions

One way to represent the uncertainty in a variable is to use a distribution. The distributions used in this thesis are triangular for their simplicity and ability to skew the median. *Triangular distributions* are often used in business simulations or project management simulations when there is only limited sample data (e.g. the relationship between variables is known but data is scarce). Formally, a triangular distribution is a continuous probability distribution with lower limit a, mode c and upper limit b with the ranges $a \in (-\infty, \infty)$, $b > a$, $a \leq c \leq b$ and probability density function (PDF):

$$PDF(x \mid a,b,c) = \begin{cases} 0 & \text{for } x < a \\ \frac{2(x-a)}{(b-a)(c-a)} & \text{for } a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & \text{for } c < x \leq b \\ 0 & \text{for } x > b \end{cases}$$

2.5.5 The ”Research to Reality” Divide

In a recent review of software effort estimation studies, Jorgensen and Shepperd have discovered a disconnect between research and the actual use of effort estimation methods [52]. They describe a, ”lack of in-depth studies on the actual use of estimation methods and real-life evaluations published as journal papers”, and warn that, ”there is a problem when there are few or no groups that conduct research on the actual use and effect of an estimation method in real-life settings [52].” Of all of the literature reviewed in their study, they could, ”find no study that had an in-depth data collection and analysis of how estimation methods were actually applied [52]!”

One of the goals of this thesis is to provide methodology improvements that are both useful and used by industrial effort estimators, and to report feedback concerning the method's real-life use.

2.5.6 Overly Optimistic Estimation

Molokken and Jorgensen report that, according to their survey, 60-80% of the time a software project will overrun its estimate by an average of 30% [98]. The 2001 Standish Group "Chaos Report" [127] shows an average overrun of 45%. This is about the same as the mean overrun of 50% reported 10 years prior by Heemstra [42]. Considering the nonparametric distribution of errors observed by our previous study [95] and discussed in section 2.5.3, these reports should consider reporting median errors and overruns. Regardless, the fact that most projects experience a cost overrun is cause for alarm.

All of these reports beg the question, "Why should it be more likely to estimate too low than too high?" Jorgensen and Grimstad [51] describe one reason, the *winner's curse*:

> "The winners curse is a result of the selection of software providers among those with the lowest bid, i.e., those with a tendency towards the highest level of overoptimism. The winners curse has not been extensively analyzed in software cost estimation studies, but is a well known phenomenon in domains such as auctioning." [51]

Unfortunately, managers may drive the budget down even further [98]. This happens because the managers may be given realistic estimates initially, but they consider the estimates too high and pressure the analysts to lower the estimates.

Finally, the initial estimate may simply be too low. It is interesting to note that one study [99] found that non-technical professionals involved in the software process provided more realistic estimates than the technical experts. Molokken and Jorgensen suggest this could be because the, "technical competence induces a bottom-up, construction-based estimation strategy", that would tend to be overly-optimistic by failing to sufficiently consider the history of previous projects [99]. Clearly, one cause for overly optimistic estimates is that the estimation model or expert judgment used doesn't consider all of the factors that influence cost. To compensate for all of these problems, it isn't uncommon for IT executives to, "get their best estimate, multiply by two, and then add a half." [127]

2.6 Feature Selection

A model is a simplification of reality. Similarly, any machine learning model must make abstractions from reality due to finite computing power. Human intuition guides this initial definition of features. Ideally, human bias wouldn't impede the algorithm's success. Unfortunately, human error is all too common, and sometimes features are selected that do not have a relationship with the target concept. This is one reason that additional features do not necessarily improve performance. The features may be poorly measured which can result in errors, or they may be redundant, needlessly adding to an algorithm's complexity. The final and most subtle reason lies in the *curse of dimensionality* which implies that, "the demand for a large number of samples grows exponentially with the dimensionality of the feature space" [28]. Feature selection is an effective tool to mitigate these problems and apply an audit, albeit limited, on human bias.

Feature selection is a technique common to machine learning domains such as data mining [135] and pattern recognition [28]. It goes by many names: feature subset selection, attribute selection, feature pruning, or with COCOMO datasets it is often called column pruning. Whatever it is called, feature selection remains the task of selecting a subset of features to use for learning while ignoring the rest in an effort to improve the algorithm's quality and efficiency. Another benefit of feature selection is that it simplifies metric collection which requires, "time and human resources and sometimes it is difficult to collect certain metrics [62]." Some feature selectors include priori knowledge from the user in their decision making process [35], but the vast majority run independently of direct human involvement. Indeed, some feature selectors use interactive visualization to guide the user in selecting attributes [107]. One of the goals of this thesis is to explore the use of feature selection for software effort estimation. Most feature selection algorithms can be classified into one of two groups: *filters* and *wrappers* [70].

2.6.1 Filters

A *filter* is a selection technique that evaluates subsets without consulting the learning algorithm [39, 135]. Instead, a filter relies on some heuristic assumption to define a feature's relevance. They are usually based on mathematical calculations such as entropy, correlation, or standard deviation. Other filters are based on a more formal definition of relevance [6]. Lashkia and Anthony argue that selecting relevant, irredundant features provides better classification accuracy than "searching for an optimal subset of attributes [73]." Filters such as Correlation-based Feature Selection(CFS) and FCBF can be very effective at eliminating redundant features [38, 39, 41, 136]. The most practical advantage of filters are that they are typically much less computationally complex than wrappers.

In addition, other machine learning algorithms may used for attribute selection [135]. For example, a decision tree might be used to select the attributes used by a nearest neighbor classifier. In this way, separate algorithms may be stacked.

2.6.2 Wrappers

A *wrapper* is a selection technique that evaluates feature subsets by their performance with the target learning algorithm [39, 135]. Although typically slower than filters, and more limited in their exploration of the attribute space, wrappers often generalize well to improved performance. Since wrappers do not ignore the effect of the features on the learning algorithm, researchers have argued that wrappers provide superior performance when compared to filters [18, 70]. Indeed, when benchmarking feature selection methods, Hall and Holmes find that the wrapper provides the best improvement in accuracy [39]. However, they also find it to be the slowest method.

By using an evaluation function based on the target learner, a wrapper designer is forced to decide how to define their training and validation datasets. Since feature selection is interested in an evaluation measure to make an internal decision and is not being used to benchmark the learner's performance, it is ok to use the same dataset for training and validation. However, this may lead to overfitting in feature selection. Any of the common cross-validation techniques, such as 10-fold cross-validation, leave-one-out cross validation, or bootstrapping, may be used [135]. Singhi and Liu report that, "in current practice separate datasets are seldom employed for selection and learning, because dividing the training data into two datasets for feature selection and classifier learning respectively reduces the amount of data that can be used in either task [123]." In their study, they conclude that, "under normal circumstances, one does not need to use separate data for feature selection and learning as recommended in the Statistics literature."

2.6.3 Feature Space

All feature selection algorithms make evaluations on instances of data. However, not all feature selection algorithms evaluate all of the feature set combinations. Given n features, there are 2^n possible subsets of features. Obviously, an exhaustive search of all 2^n subsets is only practical when n is sufficiently small, and the evaluation function sufficiently fast. Thus, suboptimal searches of the attribute space are often used.

There are many heuristics that may be used to reduce the search complexity. Filters often consider each feature in isolation, but wrappers take a more guided approach considering combinations of features. This is often done employing either a *forward selection* approach in which the search is begun with the empty set and features are added one at a time, or a *backward elimination* search is used in which the search initialized with all of the features and features may be removed one at a time [135]. Alternatively, the algorithm may employ a combination of forward selection and backward elimination to allow backtracking in an effort to avoid local minima. Some search strategies choose to evaluate each remaining candidate feature at each stage of the search. Others cut complexity by ranking the features and using this to determine which feature to consider at each stage. This allows for combinations of filters and wrappers by guiding the wrappers search with the quicker filter.

As with most search problems, popular artificial intelligence search optimization strategies such as genetic algorithms, beam search, and best-first search may all be beneficial in managing search complexity for feature selection [40, 135].

2.6.4 Feature Selection for Software Effort Estimation

Research into the application of feature subset selection techniques on software effort estimation models has been limited. A few studies by Menzies et al. have investigated the usefulness of feature selection for the COCOMO model [19, 20, 93]. In addition, feature selection has been explored for use with case-based reasoning software effort prediction models [63,64]. Alternately, in analogy-based cost estimation, instead of removing features, weights may be assigned to each feature to emphasize the better attributes [76, 132]. Menzies et al. have suggested that feature selection is useful in mitigating the deviance problem of cost estimation [90], and also found it useful in handling the instability problem of delta estimation [91](see page 35 for more on delta estimation). This is a natural reversal of the curse of dimensionality: using less features requires less instances for meaningful, stable conclusions. In light of the benefits of feature selection for effort estimation, especially considering the small datasets common to the field, there is a high demand for more in-depth research in this application.

2.7 Record Selection

Picking the correct features to use in an effort estimation model is not the only consideration to make regarding the input data. It is also useful to be critical of the data records themselves because they may suffer from the same kinds of problems as the features. In addition, depending on the novel estimation being made, not all of the records may be relevant. For example, when estimating for a very large project, very small historical projects are unlikely to be as useful as other large projects.

Thus, it is no surprise that, whereas feature selection considers subsets of features, record selection entails methods that consider subsets, or even supersets, of records for model calibration. This may be used instead of feature selection or in addition to it. The intuition behind most record selection techniques is that tuning a model with the records most relevant to the novel prediction will generalize better for improved accuracy.

There are many resampling strategies to consider. Most of these can be categorized as either *oversampling*, in which certain records are included in training multiple times, or *subsampling*, also known as undersampling, in which the training set is pruned to remove certain records. Unlike in numerical predictions like effort estimation in which resampling is done for improved regression accuracy, oversampling and undersampling are traditionally used in machine learning to balance the class variable [22]. Other strategies, such as *bootstrapping*, or sampling with replacement, will result in certain records being removed and others added multiple times. Useful record selection methods to consider for effort estimation include subsampling methods such as *stratification* and *nearest neighbor* preprocessing, and *oversampling* methods such as the use of *boosting* and *bagging*.

2.7.1 Manual Record Selection

The data used for calibration may be chosen manually. Experts may find outliers and discard those records from the model. Alternately, they may use a subset of the training data that fits some common descriptor. This is known as stratification. For example, if the training data has records from past NASA flight and ground projects, it may be best to calibrate using only the ground records when predicting for a new ground project. Stratification requires the provision of suitable labels for the training data, and imposes the bias that the distinctions chosen are relevant.

The effectiveness of using stratification for software effort estimation models is debatable [89]. Some have found the technique of great value [21, 120], while others find it offers no improvement [45]. Obviously, when making an estimate it is best to consider only relevant historical data. However, removing records that aren't quite relevant enough reduces the amount of available training data, which can be a serious problem with sufficiently small datasets.

2.7.2 Automatic Record Selection

Instead of using manual techniques, algorithms may be used that automate the selection of a subset of calibration records. These techniques use some heuristic, or rule of thumb, to select a more favorable set of calibration records.

One such rule of thumb is to select records that are similar to the novel project being predicted. When this similarity is defined by a distance metric of the records features, then a *nearest neighbor* algorithm is being employed. *Nearest-neighbor*, or *NN*, refers to a family of instance-based learning models based upon the idea that, "the properties of any particular input point x are likely to be similar to those of points in the neighborhood of x [114]. Because of this distance function, NN works best with numeric features.

In our previous work, Jalali describes a method for nearest neighbor preprocessing for the

COCOMO model [46, 95]. This method, known as LOCOMO, uses a nearest neighbor algorithm to select the training records closest to the novel record to use for calibration. This is accomplished by using a Euclidean distance function and normalizing each feature. Once the subset of training records is selected, the COCOMO algorithm is locally calibrated and used to make an estimate. LOCOMO determines the number of records to use in calibration, *k*, at run time by evaluating each possible *k* by its performance in a cross-validation experiment of the training data.

The methods of bagging and boosting, described in the next section, have much in common with automated record selection techniques because they tune the calibration records to focus on the misclassified instances. However, since these methods fundamentally change the model by combining the estimates of multiple versions of the original algorithm, they are more than simply record selection techniques.

2.8 Meta-learners

Witten and Frank describe a *meta-learner* as an algorithm that learns the best way to combine the predictions from multiple different models [135]. This definition pertains to the first combination strategy discussed in this thesis, *stacking* (section 2.8.1). However, in this report, the meaning of *meta-learner* is extended to include the techniques of *bagging* (section 2.8.2) and *boosting* (section 2.8.3) which combine predictions from different derivatives of the same model. One of the goals of this thesis is to explore the use of meta-learners for software effort estimation.

Shepperd recently conducted a study [118] to evaluate the current, ”research progress in the field of software project economics with a view to identify important challenges and promising research directions.” One of the promising areas of research suggested is the combination of prediction systems [118]. Shepperd also comments that, ”another area in which more research is needed is the less formal combining of prediction systems with expert judgement [118].”

2.8.1 Stacking

In this context, *stacking* is the class of meta-learners that combine models, ”built by different learning algorithms [135].” There are many ways that stacking may combine models:

- equal vote
- weighted vote
- use one model as a preprocessor to another model
- intelligently select the best model using some heuristic

Menzies et al. have proposed the use of stacking for software effort estimation, and offer an ”effort-modeling workbench” called COSEEKMO [93] as an example of how this can be done. COSEEKMO uses internal cross-validation and a set of heuristic ”rejection rules” to assess the results of different models. Used as a learner, COSEEKMO’s final output is the estimate from the learner it finds to perform the best. In this way COSEEKMO may reduce both the error and the variance of the estimate [93]. Recently, Jalali et al. found feature selection techniques to be more valuable than stratification using COSEEKMO [45].

2.8.2 Bagging

Bagging is a popular technique in data mining that combines the predictions of multiple models typically by using subsampling to generate many different models [135]. In bagging, each model receives an equal weight in voting, for classification problems, or in the average, for numeric problems. Another name for bagging and boosting is *arcing*, which stands for ”adaptive reweighting and combining”, and ”refers to reusing or selecting data in order to improve classification [28].” In classification problems, one advantage of meta-learners is that the classification by each learner may be examined. This may be useful to distinguish between cases of confident prediction and uncertainty. Similarly, for numeric point estimates, boosting allows for a range of estimates. However, most boosting algorithms will output some sort of weighted average, so usually this range of estimates is only viewable when looking ”under the hood” of the algorithm. For learners that already provide a range of estimates, the meta-learner may sort and combines all estimates into a single range.

Despite working by simply resampling the original training data, ”it turns out that bagging produces a combined model that often performs significantly better than a single model built from the original training data, and is never substantially worse [135].” Bagging can be very effective. For example, it greatly improves performance in outlier detection [74]. Bagging has been described as, "mainly a variance reduction (or stability) operation. [113]" In a study of machine learning techniques for software effort estimation, Braga and Oliveira found bagging to improve the performance of support vector regression, neural networks, and model trees [11]. They found the best results for the NASA dataset they used, in terms of lowest MMRE, came from using bagging with model trees [11].

2.8.3 Boosting

Boosting is, "based on the observation that finding many rough rules of thumb can be a lot easier than finding a single, highly accurate prediction rule [117]." Boosting is another meta-learner, but unlike bagging it does this using an iterative approach and combines the estimates with a weighted average [135]. Although bagging and boosting are very similar, Rosset claims, "bagging is always significantly inferior to boosting in terms of predictive performance [113]." Drucker found similar results in the regression context [27]. Another possible benefit of boosting is that by building a strong learner from a collection of weak learners, and assuming the weak learners are fast it is possible to handle complex tasks such as facial recognition faster [133]. Boosting classification learners has been shown to be a derivation of gradient descent algorithms [34, 85].

One popular boosting algorithm is the *Adaboost*, or Adaptive Boosting, algorithm [27, 28, 32, 33, 111, 114, 117, 124, 133, 135]. Adaboost works by creating a series of slightly adapted learners, or hypotheses, from its host learner. The varying hypotheses are created by adjusting the weight of each training instance's impact in calibrating the learner. At each iteration, the hypothesis generated adjusts its weights to focus on the records misclassified by the learner. In this way each hypothesis is adapted from the previous generation. The final prediction of Adaboost is the result of combining the, "probabilistic predictions", of these hypotheses [32]. In the case of a single numeric estimate this is a weighted average.

The pseudocode in from Freund and Schapire shown in figure 2.8 outlines the steps of the *Adaboost* algorithm [32]. There are many variations of the Adaboost algorithm. Freund and Schapire originally developed *Adaboost* as a 2-class classifier [32], but quickly extended it to multi-class classifiers with *Adaboost.M1* and *Adaboost.M2* [33]. *Adaboost.M1* differs from *Adaboost.M2* in

```
**Algorithm Adaboost**

Inputs :
    • sequence of $N$ labeled examples $< (x_1, y_1), \ldots, (x_N, y_N)) >$
    • distribution $D$ over the $N$ examples
    • weak learning algorithm WeakLearn
    • integer $T$ specifying number of iterations
Initialize the weight vector: $w_i^1 = D(i)$ for $1, \ldots, N$.
Do for $t = 1, 2, \ldots, T$
    • Set $p^t = \frac{w^t}{\sum_1^N w_i^t}$
    • Call WeakLearn, providing it with the distribution $p^t$; get back a hypothesis $h_t : X \rightarrow (0,1)$.
    • Calculate the error of $h_t : \varepsilon_t = \sum_1^N p_i^t |h_t(x_i) - y_i|$.
    • Set $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
    • Set the new weights vector to be $w_i^t + 1 = w_i^t \beta_t^{1-|h_t(x_i)-y_i)|}$
Output the hypothesis
    $$h_f(x) = \begin{cases} 1 & \text{if } \sum_1^T (log\frac{1}{\beta_i}) h_i(x) >= \frac{1}{2} \sum_1^T log\frac{1}{\beta_i} \\ 0 & \text{otherwise} \end{cases}$$
```

Figure 2.8: The Original Adaboost Algorithm [32]

that the *M1* version outputs a single class, but the *M2* version outputs a set of possible classes with a degree of plausibility for each. They also gave suggestions on how the algorithm might be tailored for regression problems. Drucker was the first to modify Adaboost for regression [27], and later Ridgeway et al. created *Adaboost.R* [111] from Freund and Schapire's vision which also modified for regression problems. More variants of Adaboost for regression problems evolved from *Adaboost.R*, including *Adaboost.RA* [7], and *Adaboost.RT* [124].

Although Adaboost appears to only work with algorithms that can handle weighted training data, this is in fact a false constraint [33]. For algorithms that use unweighted examples for learning, Adaboost can be used by sampling at random with replacement from a weighted distribution. As the size of this set approaches infinity, the effective unweighted training set is equivalent to the weighted set.

2.9 Summary of Related Work

There are many methods for software effort estimation. Jorgensen [49, 52] found that although the most popular estimation method in industry is expert-judgment, the research is dominated by model-based methods. Of these model-based methods, 51% of the research from 2000 to 2004 concerned regression techniques.

A recent study [118] by Shepperd found one of the promising areas of new research in software effort estimation to be the combination of prediction systems. Menzies et al. [93] have created one such combination system, COSEEKMO, which compares different estimation models using cross-validation and "rejection rules", and finally uses the model that performs the best.

Shepperd's study particularly suggests new research in, "the less formal combining of prediction systems with expert judgement [118]." Similarly, Meli [86] has suggested an integrated approach combining expert judgment with models:

> "People and methods are tremendously important to gain a high quality in the estimation results. Instead of conceiving them as alternative means we should consider them as synergic resources. This may happen if we are supported by a conceptual framework of integration and a set of operational rules to follow." [86]

Jorgensen [49] has documented twelve best practices for expert-based estimation, and Menzies et al. [93] have extended this list to include other best practices useful for software effort estimation in general.

Research has suggested feature and record selection techniques can be useful for software effort estimation models. In addition, machine learning techniques such as bagging and boosting could lead to stabler, more accurate estimates.

Unfortunately, software effort estimation is plagued by many problems including highly inaccurate and uncertain estimates which are often overly optimistic, small amounts of historical data, and research that is often separated from the industrial reality and offers inconsistent results. Fortunately, our previous work [95] has discovered the inconsistent results come from the non-Gaussian estimate error distributions, and show that nonparametric evaluation methods can provide more consistent results. In addition, Kitchenham [68] has described the sources of estimate uncertainty.

The existing literature helped inspire the following research goals for this thesis:

1. provide methodology improvements that are both useful and used by industrial effort estimators, and to report feedback concerning the method's real-life use

2. represent estimation uncertainty by accounting for Kitchenham's sources of estimation error [68]

3. reduce the uncertainty of the estimate

4. make progress towards the, ”conceptual framework of integration” of expert and model-based methods suggested by Meli [86]

5. implement the expert-judgment ”best practices” proposed by Jorgensen [49] in a model

6. provide estimation methods useful at different software life-cycle stages

7. improve estimation accuracy

8. explore the use of feature selection for software effort estimation

9. explore the use of meta-learners such as bagging and boosting for software effort estimation

10. evaluate methods using the nonparametric evaluation discussed in our previous work [95]

Chapter 3

Laboratory Studies

The majority of the early work of this thesis is presented in this chapter which explores some effort estimation methods that had not been tried before. Unfortunately, the results were somewhat disappointing because the supposedly more sophisticated and complex methods failed to make improvements. Actually the very simple methods did as well as anything else, as can be found later in this chapter. The results of these experiments helped guide the inclusion of methods in a new estimation framework discussed in Chapters 4 and 5.

This chapter details some laboratory studies into the use of feature subset selection (section 3.2), bagging (section 3.3), and boosting (section 3.4) on the COCOMO software effort estimation model. COCOMO was chosen over the other estimation methods described in Chapter 2 for several reasons: First, COCOMO's implementation is straightforward and openly described by Boehm [8, 10]. Second, software effort estimation data is hard to come by and COCOMO data was available. Third, abundant research on COCOMO was available. In addition, the COCOMO II model, "has been validated using JPL data and provides reasonable flight and ground software estimates in the JPL environment [78]." Finally, it was chosen because these experiments are part of a larger study using COCOMO including our prior work [95] and the work of Jalali [46].

3.1 Experimental Design

All of the algorithms explored in this study are based upon either Boehm's COCOMO model. The experiments discussed in this chapter were based up the 1981 version of COCOMO [8] due to the available data. The case study experiment in the next chapter is based upon Boehm's updated COCOMO II model [10].

3.1.1 Data Used in This Study

In this study, effort estimators were built primarily using all or some *part* of data from the following two sources:

*COC*81*:* 63 records in the COCOMO-I format. Source: [8, p496-497]. Download from `http://unbox.org/wisp/trunk/cocomo/data/coc81modeTypeLangType.csv`.

*NASA*93*:* 93 NASA records in the COCOMO-I format. Download from `http://unbox.org/wisp/trunk/cocomo/data/nasa93.csv`.

Taken together, these two sets are the largest COCOMO-style data source in the public domain (for reasons of corporate confidentiality, access to Boehm's COCOMO-II data set is highly restricted). *Coc*81 is the original COCOMO data used by Boehm to calibrate COCOMO 81. Funded by the Space Station Freedom Program, *NASA*93 was originally collected to create a NASA-tuned version of COCOMO. It contains data from the Jet Propulsion Laboratory and five other NASA centers. Thus, it "covers a wide range of software domains, development processes, languages, and complexity as well as fundamental differences in culture and business practices between each center [93]." The *Nasa*93 dataset "has been in the public domain for several years but few have been aware of it [93]." Although both of these datasets are dated, they are still useful for comparing different estimation methods.

> "When the *nasa*93 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 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." [93]

Menzies et al. have observed wider error variances with the *NASA*93 dataset than in the *COC*81 dataset [93]. They suspect this is "due to the wide variety of projects in that data set, and *not* poor data collection", and argue that, "*nasa*93 was collected using methods equal to, or better, than standard industrial practice" [93].

Different subsets and number of subsets used (in parenthesis) are:

All(2): selects all records from a particular source.

Category(2): *NASA*93 designation selecting the type of project; e.g. avionics.

Center(2): *NASA*93 designation selecting records relating to where the software was built.

Fg(1): *NASA*93 designation selecting either "*f*" (flight) or "*g*" (ground) software.

Kind(2): *COC*81 designation selecting records relating to the development platform; e.g. max is mainframe.

Lang(2): *COC*81 designation selecting records about different development languages; e.g ftn is FORTRAN.

Mode(4): designation selecting records relating to the COCOMO-I development mode: one of semi-detached, embedded, and organic.

Project(2): *NASA*93 designation selecting records relating to the name of the project.

Year(2): is a *NASA*93 term that selects the development years, grouped into units of five; e.g. 1970, 1971, 1972, 1973, 1974 are labeled "1970".

There are more than 19 subsets overall. Some have fewer than 20 projects and hence were not used. The justification for using 20 projects or more is offered in [93].

One other dataset was used in this study besides *NASA*93 and *COC*81. This additional dataset, *NASA*07, was used in an experiment for NASA's Jet Propulsion Laboratory to evaluate the effectiveness of various effort estimation strategies. This dataset, although not public, was of key importance because it contained an updated training dataset and a distinctly separate test dataset that represents the types of projects that NASA JPL frequently estimates. Thus, *NASA*07 was useful to evaluate the methods for NASA JPL's domain. More on this dataset and the experiment using it can be found in section 5.3.

3.1.2 Method Evaluation

Cross-validation methods are discussed in section 2.5.3. In this study, leave-one-out cross-validation, also known as *n*-fold cross-validation, was chosen for several reasons. First, the datasets in question are small and thus the computationally expensive drawback of *n*-fold was minimized. Second, this technique may provide the most accurate evaluations. Witten and Frank comment that, "leave-one-out seems to offer a chance of squeezing the maximum out of a small dataset and obtaining as accurate an estimate as possible [135]." Third, since *n*-fold does not require randomization the experiments are easily repeatable by other parties, and a random number generator seed does not need to be stored. Finally, this method provides the training algorithms with the maximum amount of data which may be important considering the small datasets.

The methods were evaluated using both the nonparametric techniques proposed in our previous work [95], and standard parametric techniques. The nonparametric evaluation, a Mann Whitney Wilcoxon rank-sum test at a 95% confidence level, is given priority due to the non-Gaussian error distributions seen in software effort estimation models [95]. The parametric evaluations are included for three reasons. First, in the case of ties using the nonparametric methods, the parametric measures may give insight into which method may be best. Unfortunately, this is not statistically sound or the method would've won the nonparametric evaluation. Nevertheless, it may be a useful hint. Second, the parametric measures may be preferred by readers that are skeptics or new to the nonparametric methods. Third, the parametric scores allow the algorithms to be compared to other methods outside of the given experiment with the same or similar data whereas the nonparametric evaluation has to be rerun again. More on method evaluation may be found in section 2.5.3.

The evaluation scripts used in this study are available online at `http://unbox.org/wisp/var/dan/eval/`.

When considering the methods, emphasis is placed on the method with the minimum losses, maximum wins-losses, and secondarily having a low median MRE.

3.2 Feature Subset Selection

This section discusses some algorithms and experiments to investigate feature subset selection, FSS, algorithms for software effort estimation, which happens to be one of the goals of this thesis. For more on FSS in the literature, see section 2.6.

Each of the feature selectors described are sometimes used in the context of a pure feature selector, and other times as a learner. In fact they may be both. They may be used to simply prune the feature set and then hand this off to another algorithm. When used in the context of a learner, the implied meaning is that the feature selection is used, and COCOMO calibrated with local calibration is used with this restricted feature set.

3.2.1 Cocomost

FSS algorithms can be described by two characteristics: their evaluation method and their search strategy. The evaluation method is how they determine whether a given subset of features is better than another. The search strategy is how they approach the problem of evaluating the possible subsets of features. For n features, there are 2^n possible subsets to consider. Most FSS algorithms won't try all of these subsets but will instead use heuristics to search a subset of them. However, with a small dataset with a relatively small amount of features and a quick evaluation step, an exhaustive search may still be possible. The first algorithm investigated, Cocomost, evaluates the complete set of possible feature subsets.

Cocomost is an FSS algorithm for the COCOMO 81 model that evaluates all of the possible subsets of features. A variety of evaluation methods were tried, and each was evaluated by training and testing on the training data. The evaluation methods considered included correlation, median MRE, MMRE, standard deviation of MRE, various PRED measures, as well as some custom measures, MMRE-A-to-B. This custom measure MMRE-A-to-B will sort the errors, and return the average MRE between the A and B percentiles of the sorted list. For example, if A is 10 and B is 90 then this is essentially MMRE ignoring the lowest 10% and highest 10% of the data. The inspiration for this measure was to lessen the impact of outliers on the evaluation. The other listed evaluation criteria are described in section 2.5.3.

The code used in this experiment is available online at `http://unbox.org/wisp/var/dan/cocomost/`.

The Cocomost algorithm was run on the 19 datasets described in section 3.1.1 using *n*-fold cross-validation. The results over all of the datasets are shown in table 3.1. Similarly, the results on just the Coc81 data are in table 3.2, and for Nasa93 data in table 3.3.

Over all of the datasets, table 3.1 shows basic LC, which is COCOMO trained to the local data using all of the features, never lost a nonparametric test. However, some of the Cocomost methods did have a higher wins-losses score. There is no significant difference in median MRE. The Cocomost variant that evaluated using standard deviation did the worst having the most losses and worst median MRE of any method. However, it still had a lot of ties and the median MRE was only about 5% higher so it didn't do terrible.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
ExhaustiveFSS_pred25	220	7	1	6	57.01	28.97	0.52	0.794
ExhaustiveFSS_mmre	221	6	1	5	54.42	26.88	0.51	0.824
LC	225	3	0	3	55.38	26.82	0.53	0.848
ExhaustiveFSS_mmre20to80	223	4	1	3	55.53	27.18	0.51	0.799
ExhaustiveFSS_pred30	226	2	0	2	56.68	27.87	0.52	0.794
ExhaustiveFSS_mmre25to75	226	2	0	2	56.36	27.96	0.51	0.790
ExhaustiveFSS_median_mre	227	1	0	1	58.19	28.22	0.50	0.786
ExhaustiveFSS_mmre10to90	225	2	1	1	54.51	28.30	0.50	0.804
ExhaustiveFSS_pred40	225	1	2	-1	54.71	28.56	0.52	0.807
ExhaustiveFSS_mmre33to67	222	2	4	-2	56.08	26.92	0.52	0.802
ExhaustiveFSS_corr	222	2	4	-2	55.87	29.63	0.50	0.832
ExhaustiveFSS_pred50	221	1	6	-5	54.52	27.54	0.52	0.823
ExhaustiveFSS_sd(mre)	213	1	14	-13	59.17	31.54	0.47	0.781

Table 3.1: Cocomost FSS Experiment Results from All Data

Using the Coc81 data, table 3.2 shows LC had a better median MRE than all other methods, and it had minimum losses and maximum wins-losses. In this case, the feature selection methods deteriorated performance.

The results were more promising using the Nasa93 data (see table 3.3). Although LC did not lose, neither did some of the other methods. In addition, some of the other methods had

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	83	1	0	1	41.16	27.46	0.49	0.915
ExhaustiveFSS_corr	84	0	0	0	50.19	36.69	0.40	0.850
ExhaustiveFSS_median_mre	84	0	0	0	55.89	34.28	0.41	0.758
ExhaustiveFSS_mmre	84	0	0	0	45.25	32.30	0.42	0.862
ExhaustiveFSS_mmre10to90	84	0	0	0	46.72	33.40	0.40	0.812
ExhaustiveFSS_mmre20to80	84	0	0	0	49.97	33.68	0.41	0.810
ExhaustiveFSS_mmre25to75	84	0	0	0	52.72	36.66	0.41	0.791
ExhaustiveFSS_mmre33to67	84	0	0	0	51.72	32.12	0.44	0.812
ExhaustiveFSS_pred30	84	0	0	0	53.77	33.19	0.42	0.797
ExhaustiveFSS_pred40	84	0	0	0	48.98	30.97	0.46	0.803
ExhaustiveFSS_pred50	84	0	0	0	48.43	30.96	0.46	0.838
ExhaustiveFSS_sd(mre)	84	0	0	0	47.84	33.74	0.42	0.874
ExhaustiveFSS_pred25	83	0	1	-1	54.28	37.65	0.41	0.789

Table 3.2: Cocomost FSS Experiment Results from Coc81 Data

better wins-losses scores and lower median MRE's. The median MRE improvements were slight but noticeable, at about 4%. However, due to the large number of ties, no clear winner may be declared.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
ExhaustiveFSS_pred25	137	7	0	7	58.60	23.90	0.58	0.797
ExhaustiveFSS_mmre	137	6	1	5	59.77	23.72	0.57	0.802
ExhaustiveFSS_mmre20to80	139	4	1	3	58.78	23.38	0.57	0.792
ExhaustiveFSS_pred30	142	2	0	2	58.37	24.76	0.58	0.793
LC	142	2	0	2	63.67	26.44	0.56	0.809
ExhaustiveFSS_mmre25to75	142	2	0	2	58.48	22.88	0.58	0.789
ExhaustiveFSS_median_mre	143	1	0	1	59.54	24.69	0.55	0.803
ExhaustiveFSS_mmre10to90	141	2	1	1	59.06	25.32	0.56	0.799
ExhaustiveFSS_pred40	141	1	2	-1	58.06	27.16	0.55	0.810
ExhaustiveFSS_mmre33to67	138	2	4	-2	58.62	23.89	0.56	0.796
ExhaustiveFSS_corr	138	2	4	-2	59.18	25.51	0.57	0.822
ExhaustiveFSS_pred50	137	1	6	-5	58.08	25.55	0.56	0.815
ExhaustiveFSS_sd(mre)	129	1	14	-13	65.78	30.25	0.49	0.726

Table 3.3: Cocomost FSS Experiment Results from Nasa93 Data

3.2.2 dBFS

The biggest drawback of an exhaustive search is that each additional feature in the initial feature set doubles the number of subsets to consider. Clearly this approach does not scale well. While Cocomost is fast enough for COCOMO 81, it does not scale well to COCOMO II which has more cost drivers.

Thus, suboptimal searches of the attribute space that use heuristics to make an educated guess as to where the optimal solution exists are employed. This section as well as section 3.2.3 describe some experiments using suboptimal feature selectors. In this work, many roads were tried, few got anywhere, and most were dead-ends. However, in the end, some paths using a suboptimal search were found that performed as well as the exhaustive search.

Popular suboptimal subset searches include the forward selection and backward elimination search. In a forward selection search, the algorithm starts with an empty feature set (except for KLOC for COCOMO), and tries adding a feature one at a time. If it improves performance the feature is kept. Conversely, a backward elimination search starts with all of the features and tries throwing them away one at a time. The traditional way to use these algorithms is at each step to evaluate the effect of each remaining feature to consider, and choose the one with the best impact. For example, in a backward elimination search, each feature is considered for elimination and the one that when absent improves the performance the most is chosen. Then this is done again with the remaining features until a preset m number of features is obtained.

In this study the backward elimination search was chosen to use, but was built so that it did not make an assumption as to the number of features the optimal subset should contain. Thus, the approach is called a dynamic backward elimination search, or *dBFS*. The forward selection search was not tried because early results from the next section showed backward elimination performed better.

The number of subset evaluations to perform for the dBFS algorithm is $\sum_{i=1}^{n} i$ where n is the number of features to consider.

The dBFS algorithm was tested using the same types of evaluation criteria as Cocomost. However, a new idea was tried as well. This new idea is denoted in the results tables by a number at the end of the method name. This number is a threshold that the performance of the new subset being tested minus the old performance must beat for the feature to be thrown away. The inspiration for this idea is that perhaps a feature should only be thrown away if it greatly decreases performance to avoid throwing away features carelessly. Conversely, the threshold could be set to be negative such that features are only kept if they are truly necessary.

The code used in this experiment is available online at `http://unbox.org/wisp/var/dan/dBFS/`.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
dBFS_mmre_0.000	2221	59	0	59	54.22	26.67	0.53	0.829
LC	2250	30	0	30	55.38	26.82	0.53	0.845
dBFS_corr_-0.005	2123	10	147	-137	56.19	30.23	0.49	0.841

Table 3.4: Selected dBFS Results from All Data

Table 3.4 shows some selected results from the dBFS experiment using all of the data. The rest of the experiment's results are freely available at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`. The results from the dBFS experiment were not much different from Cocomost: LC held its own while some of the dBFS results were the same.

In addition, LC once again dominated in the Coc81 dataset (table 3.5), while the FSS methods showed some promise with the Nasa93 dataset (table 3.6). The dBFS method evaluating with MMRE and the default threshold of 0 had more wins than LC and a slightly better median MRE. The MMRE is also better, but as we have shown [95], this is not a valid statistic for software effort estimation models.

The threshold concept did not appear to have an effect. This is not a big surprise considering the small datasets and that in many cases the threshold will not change the algorithm's decision.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	830	10	0	10	41.16	27.46	0.49	0.914
dBFS_mmre_0.040	830	10	0	10	44.58	31.68	0.43	0.872
dBFS_corr_0.000	819	0	21	-21	50.32	36.40	0.39	0.860

Table 3.5: Selected dBFS Results from Coc81 Data

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
dBFS_mmre_0.000	1391	49	0	49	59.85	23.75	0.58	0.805
LC	1420	20	0	20	63.67	26.44	0.56	0.805
dBFS_corr_-0.005	1304	10	126	-116	59.62	26.63	0.55	0.829

Table 3.6: Selected dBFS Results from Nasa93 Data

3.2.3 Cocomin

Cocomin is a near linear-time pre-processor that selects the features on some heuristic criteria and does not explore all subsets of the features. It runs in $O(F \cdot log(F))$ for the sort and $O(F)$ time for the exploration of selected features.

The concept behind *Cocomin* was to create a very fast feature selector. This was inspired by some previous algorithms that were very slow. While the biggest gains in speed of all of the feature selectors in this thesis came from using efficient C code, the theoretical interest for Cocomin is its very small footprint on the attribute space.

Cocomin is defined by the following operators:

$$\{sorter, order, learner, scorer\}$$

The algorithm runs in linear time over a *sorted* set of features, F.

Cocomin pre-sorts the features on some heuristic ranking. Some of these criteria, such as standard deviation or entropy, are gathered without evaluation of the target learner. Others are gathered by evaluating the performance of the learner using only the feature in question plus any required features, such as KLOC for COCOMO, to calibrate the model.

This search can be *order*ed in one of three ways:

- A “backward elimination” process starts with all features F and throws some away, one at a time.
- A “forward selection” process starts with one feature and adds in the rest, one at a time.
- “Both” forward and backward searches are run separately and the best performing result is chosen.

Regardless of the search order, at some point the current set of features $F' \subseteq F$ is passed to a *learner* to generate a performance *score* by applying the model learned on the current features to the $Train$ set. Cocomin returns the features associated with the highest score.

After the features are ordered, each feature is considered for backward elimination, or forward selection if chosen, in a single linear pass through the feature space, F. The decision to keep or discard the feature is based on an evaluation measure generated by calibrating and evaluating the model with the training data.

The variations of Cocomin tried the following ranking measures:

- MMRE (High and Low)
- Median MRE (High and Low)
- Correlation (High and Low)
- Pred30 (High and Low)
- Native (The order was the in the default order of the data)
- Random
- Entropy (High and Low)
- Variance of MRE (High and Low)

The first four ranking measures are created by running an experiment with the training data using only the attribute in question plus KLOC and returning the relevant performance score. The remaining four ranking techniques are calculated using only the data without running the target learner. Thus they are filter methods and are faster than the first four which are wrapper techniques.

Entropy is a measure of information content in a signal. It can be calculated directly from the data without invoking an internal evaluation of the target learner. For a set of n numbers $X_i ... Xn$, the entropy of this set of numbers is given by $Entropy(X) = \sum_{i=1}^{n} p(X_i) * logp(X_i)$ [135]. The p function is simply a normalization function that divides each element in the equation by the sum of X. In this study the natural logarithm was used.

The method name identifying each Cocomin variant in the results tables follows the following schema: cocomin81_SEARCH_RANK_EVAL where *SEARCH* is either forward, backward, or both, *RANK* is the method used to presort the features, and *EVAL* is the method used to determine whether one feature subset is better than another.

The code used in this experiment is available online at `http://unbox.org/wisp/var/dan/experiment_cocomin81/`.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	6572	610	0	610	55.38	26.82	0.53	0.845
cocomin81_both_med_mre_hi_corr	6572	610	0	610	55.38	26.82	0.53	0.847
cocomin81_both_med_mre_hi_mmre	6645	537	0	537	53.30	26.75	0.51	0.820
cocomin81_backward_med_mre_hi_mmre	6647	535	0	535	53.41	27.05	0.52	0.821
cocomin81_forward_mmre_lo_mmre	6699	483	0	483	54.96	26.03	0.52	0.804
cocomin81_forward_cor_hi_corr	5491	0	1691	-1691	80.64	44.93	0.38	0.576

Table 3.7: Selected Cocomin Results from All Data

The rest of the experiment's results are freely available at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`. Over all of the data (illustrated by table 3.7), Cocomin was able to did as well as basic LC. It is possible it adds slight improvement although this is hard to tell. However, once again the feature selectors failed to offer improvement on the Coc81 data (see table 3.8), but offered some improvement on the Nasa93 data (see table 3.9). The best performing Cocomin variant on the Nasa93 data tried both a backward and forward search, pre-sorted on lowest MMRE, and evaluated using MMRE. It had the same number of losses as LC, which was 0. It had 127 more wins out of 4536 total and a median MMRE that was 3.54% lower. Thus, for this data, Cocomin offers as good as or slightly better accuracy than standard LC. It also provides the other benefits of FSS including reducing uncertainty (stabler estimates) by reducing the effects of human error. It also creates a simpler, faster model.

3.2.4 Comparing Each FSS Method

One startling result of this study was that an exhaustive search of the attribute space, i.e. Cocomost, performed about the same as a very limited search of the feature space, i.e. Cocomin. This is an exciting discovery because it provides a very simple and efficient feature selector without

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	2245	401	0	401	41.16	27.46	0.49	0.913
cocomin81_backward_cor_hi_corr	2245	401	0	401	41.16	27.46	0.49	0.913
cocomin81_both_med_mre_hi_corr	2245	401	0	401	41.16	27.46	0.49	0.913
cocomin81_both_med_mre_hi_mmre	2396	250	0	250	44.67	31.91	0.43	0.851
cocomin81_both_mmre_lo_mmre	2418	228	0	228	45.69	32.39	0.41	0.864
cocomin81_forward_cor_lo_pred30	1544	0	1102	-1102	80.95	53.53	0.27	0.622

Table 3.8: Selected Cocomin Results from Coc81 Data

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
cocomin81_both_mmre_lo_mmre	4200	336	0	336	58.79	22.90	0.58	0.801
cocomin81_backward_pred_hi_mmre	4208	328	0	328	58.91	23.19	0.57	0.792
cocomin81_forward_mmre_lo_mmre	4220	316	0	316	58.39	23.61	0.57	0.786
LC	4327	209	0	209	63.67	26.44	0.56	0.806
cocomin81_both_med_mre_hi_corr	4327	209	0	209	63.67	26.44	0.56	0.808
cocomin81_forward_mmre_hi_median_mre	3651	0	885	-885	74.06	37.75	0.44	0.584

Table 3.9: Selected Cocomin Results from Nasa93 Data

sacrificing quality.

How can a very limited search be as good as an exhaustive search? There are a few explanations for this phenomenon. The first is that the available data generalizes very poorly to predictions of unseen data. This hypothesis would mean that the feature selector is essentially wandering in the dark and its results are more or less random. The second explanation is that the small dataset results in a fair amount of noise making a truly optimal solution impossible to find, and thus suboptimal solutions are as good as it gets. The final explanation is that the Cocomin algorithm's guided search is very effective, and so an exhaustive search is truly unnecessary.

Whatever the cause of these results, the effect is that the current results show Cocomin might as well be used because it is the fastest method and performs just as well as the other feature selectors.

To further verify this, an experiment was done including LC, Cocomost, dBFS, and Cocomin so that the nonparametric evaluation could compare them to each other. The results shown in tables 3.10, 3.11, and 3.12 confirm that the methods perform about the same. It is interesting to note that Cocomin has the lowest median MRE of all, however the other feature selectors have such similar values that the difference is negligible.

The code used in this experiment is available online at `http://unbox.org/wisp/var/dan/`

fss/.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
ExhaustiveFSS_mmre	57	0	0	0	54.42	26.88	0.51	0.826
LC	57	0	0	0	55.38	26.82	0.53	0.848
cocomin81_both_mmre_lo_mmre	57	0	0	0	53.96	26.40	0.52	0.824
dBFS_mmre_0.000	57	0	0	0	54.22	26.67	0.53	0.829

Table 3.10: Experiment Comparing FSS Methods on All Data

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
ExhaustiveFSS_mmre	21	0	0	0	45.25	32.30	0.42	0.868
LC	21	0	0	0	41.16	27.46	0.49	0.915
cocomin81_both_mmre_lo_mmre	21	0	0	0	45.69	32.39	0.41	0.865
dBFS_mmre_0.000	21	0	0	0	44.58	31.68	0.43	0.870

Table 3.11: Experiment Comparing FSS Methods on Coc81 Data

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
ExhaustiveFSS_mmre	36	0	0	0	59.77	23.72	0.57	0.802
LC	36	0	0	0	63.67	26.44	0.56	0.809
cocomin81_both_mmre_lo_mmre	36	0	0	0	58.79	22.90	0.58	0.799
dBFS_mmre_0.000	36	0	0	0	59.85	23.75	0.58	0.804

Table 3.12: Experiment Comparing FSS Methods on Nasa93 Data

3.3 Bagging

The concept of bagging, described in section 2.8.2, may be of great benefit to software effort estimation. Three experiments were run to investigate the use of bagging on the COCOMO model, which is one of the goals of this thesis.

3.3.1 1st Bagging Experiment

The first bagging experiment was run to create a number of virtual experts, each with a slightly different knowledge of the target concept. When making an estimate, each expert separately makes an estimate, and the results are combined. This is done using either an average or a weighted average based on the expert's performance. Both average vote and weighted average were tried in this study. The weighting coefficient used was the inverse of the expert's MMRE from its estimates on the training data. In the results tables this is denoted by ”use_weight:” followed by either a 0 for average vote or 1 for weighted vote.

The experts are created by calibrating using different subsets of the data. Several different amounts of expert *knowledge*, or the percent of the data used for calibration, was tried in this experiment. In the results tables this is denoted by ”exp_know:” followed by a number ¡ 1 representing the percent of the data used. In addition, different numbers of experts were tried, and this is denoted by ”exp_num:” followed by the respective number of experts in the results tables.

Each expert was calibrated on its knowledge base using the Cocomin algorithm described above. The primary inspiration for this choice was that the differences in calibration data among experts could result in slightly different feature sets chosen by Cocomin, and this additional variety among experts should be good for the algorithm. After all, if all of the experts are the same there is no reason to have more than one.

This experiment used a version of Cocomin, which at the time was considered the best, that used the native ordering of the features, and ran a backward-elimination search pruning features using MMRE.

The code used in this experiment is available online at `http://unbox.org/wisp/var/dan/bagging/experiment1/`.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
use_weight:0_exp_num:14_exp_know:0.30	9398	7	0	7	56.42	27.82	0.52	0.819
use_weight:1_exp_num:10_exp_know:0.50	9400	5	0	5	56.98	27.29	0.52	0.829
LC	9404	1	0	1	55.38	26.82	0.53	0.846
use_weight:1_exp_num:2_exp_know:0.35	9394	0	11	-11	60.84	27.66	0.51	0.803

Table 3.13: Selected Results from 1st Bagging Experiment on All Data)

The results of this first bagging experiment are illustrated in tables 3.13, 3.14, and 3.15, and the full results are freely available at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`.. These results showed no significant difference over standard LC. Once again the results were worse with the Coc81 data than the Nasa93 data, but this can likely be attributed to the use of Cocomin in the bagging collection. Any improvements from the first bagging algorithm tried is highly marginal at best, and thus not worth the additional computational complexity.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	3464	1	0	1	41.16	27.46	0.49	0.912
use_weight:1_exp_num:8_exp_know:0.90	3464	1	0	1	43.78	31.40	0.46	0.879
use_weight:1_exp_num:2_exp_know:0.35	3454	0	11	-11	52.14	34.21	0.39	0.824

Table 3.14: Selected Results from 1st Bagging Experiment on Coc81 Data)

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
use_weight:0_exp_num:14_exp_know:0.30	5933	7	0	7	61.64	24.65	0.57	0.800
use_weight:1_exp_num:17_exp_know:0.60	5940	0	0	0	59.60	22.49	0.59	0.807
LC	5940	0	0	0	63.67	26.44	0.56	0.808
use_weight:0_exp_num:2_exp_know:0.30	5936	0	4	-4	64.92	28.26	0.52	0.753

Table 3.15: Selected Results from 1st Bagging Experiment on Nasa93 Data)

3.3.2 2nd Bagging Experiment

The second bagging experiment was the same as the first except it used a random pre-sorting routine in its COCOMIN feature selector in order to increase the difference between each expert.

The results of this experiment are illustrated in tables 3.16, 3.17, and 3.18, and are fully shown at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`. These results were basically the same as in the 1st bagging experiment with LC, and the bagging learners performed roughly equivalently. The differences in terms of wins, losses, and median MRE are mostly negligible. For example, on the Nasa93 dataset, a variation of bagging (see table 3.18) using 10 experts with 35% knowledge and a weighted average combination scheme produced a 4.7% reduction in median MRE. This may be a slight improvement, although the noise in the data makes it hard to be sure. The most impressive result from this is that experts with only 35% knowledge of a small dataset performed this well as a group.

The code used in this experiment is available online at `http://unbox.org/wisp/var/dan/bagging/experiment2/`.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
use_weight:1_exp_num:6_exp_know:0.45	9399	6	0	6	55.37	27.21	0.53	0.825
use_weight:0_exp_num:8_exp_know:0.55	9405	0	0	0	54.90	24.98	0.55	0.811
LC	9405	0	0	0	55.38	26.82	0.53	0.846
use_weight:1_exp_num:3_exp_know:0.35	9404	0	1	-1	58.91	29.53	0.50	0.824

Table 3.16: Selected Results from 2nd Bagging Experiment on All Data)

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	3465	0	0	0	41.16	27.46	0.49	0.912
use_weight:1_exp_num:2_exp_know:0.40	3465	0	0	0	45.13	28.34	0.50	0.862
use_weight:0_exp_num:11_exp_know:0.35	3465	0	0	0	47.89	36.65	0.39	0.825

Table 3.17: Selected Results from 2nd Bagging Experiment on Coc81 Data)

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
use_weight:1_exp_num:6_exp_know:0.45	5934	6	0	6	60.75	24.41	0.57	0.810
use_weight:1_exp_num:10_exp_know:0.35	5940	0	0	0	61.10	21.75	0.61	0.811
LC	5940	0	0	0	63.67	26.44	0.56	0.808
use_weight:1_exp_num:2_exp_know:0.35	5938	0	2	-2	65.52	26.92	0.53	0.795

Table 3.18: Selected Results from 2nd Bagging Experiment on Nasa93 Data)

3.3.3 3rd Bagging Experiment: Cocomonster

The 3rd bagging experiment tried involved the creation of an algorithm that creates a learner using every permutation of the feature set. Each of these learners is evaluated by making estimates on the training data. The final estimate produced is the weighted average of the estimate from each expert, weighted by the inverse of their MMRE. This algorithm was named *Cocomonster* for its ridiculous computational complexity.

The code used in this experiment is available online at `http://unbox.org/wisp/var/dan/bagging/cocomonsterV1/`.

There is some kind of intrinsic appeal to overly complex algorithms. One may think, "It must be better; it's just so...complicated." This is much like the snob factor in economics which leads us to believe that a diamond must be rare and valuable because it costs a lot of money. This simply isn't always true, and the results show it (see tables 3.19, 3.20, and 3.21).

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	17	2	0	2	55.38	26.82	0.53	0.848
cocomonster	17	0	2	-2	63.39	33.88	0.44	0.750

Table 3.19: Cocomonster Bagging Results from All Data

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	5	2	0	2	41.16	27.46	0.49	0.915
cocomonster	5	0	2	-2	62.10	41.20	0.33	0.765

Table 3.20: Cocomonster Bagging Results from Coc81 Data

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	12	0	0	0	63.67	26.44	0.56	0.809
cocomonster	12	0	0	0	64.14	29.60	0.51	0.742

Table 3.21: Cocomonster Bagging Results from Nasa93 Data

Perhaps the reason the Cocomonster algorithm performed poorly is the inclusion of many sub-optimal experts generated by the exhaustive combination of feature set permutations. A similar algorithm that cut off combinations believed to only hinder performance may produce better results.

3.4 Boosting

Boosting, described in section 2.8.3, has the potential to improve the accuracy of software effort estimation models. This section describes some algorithms developed using boosting with the COCOMO algorithm, and includes results from experiments involving these algorithms. The investigation of boosting for software effort estimation is one of the goals of this thesis.

There are several ways that boosting algorithms may manipulate their calibration dataset in order to create successively different derivations of the base learner that are guided towards the most difficult to predict instances. This may be done using subsampling, oversampling, or distributive sampling. Each of these approaches was applied and the results are described in the following subsections.

3.4.1 Boosting by Subsampling

The subsampling approach to boosting calibrates using proper subsets of the original data such that the focus is placed upon the instances most difficult to classify.

The boosting by subsampling experiment tried boosting both Cocomin as well as standard LC. These boosting algorithms may be described by their base learner, either Cocomin or LC, the number of boosting iterations, and the subsample fraction used by each iteration. The Cocomin variant used evaluated using MMRE, used a native ordering of the attributes, and used a backward-elimination search.

This boosting algorithm's final prediction is the result of using the hypothesis from each boosting iteration to make a prediction, and then combining these estimates using a weighted average where the weight coefficient is the inverse of the MMRE of an evaluation on the training data.

The code used in this experiment is available online at `http://unbox.org/wisp/var/dan/boosting/subsampling/`.

The subsampling results show some promise (illustrated in tables 3.22, 3.23, and 3.24, and are shown in full at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`). Similar to the other experiments, LC did not lose, but neither did many of the other methods. Over all of the data, the boosted Cocomin showed about a 5% reduction in median MRE. Contrary to many other experiment's results, the performance did not slightly degrade on the Coc81 data, but rather slightly

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
Cocomin_Boost:5_Subsample:0.40	5691	142	0	142	54.47	26.26	0.54	0.839
LC_Boost:5_Subsample:0.95	5898	106	0	106	59.77	29.00	0.57	0.914
LC_Boost:20_Subsample:0.90	5898	106	0	106	59.75	28.91	0.57	0.915
Cocomin_Boost:10_Subsample:0.75	5343	109	4	105	48.35	23.78	0.48	0.742
LC	5898	106	0	106	59.91	28.96	0.57	0.914
Cocomin	5906	97	1	96	58.18	29.04	0.56	0.882

Table 3.22: Selected Boosting by Subsampling Results on All Data)

improved, although this difference is a small and arguable. On the Nasa93 data, one of the boosted Cocomin variants reported a median MRE about 11% lower than standard LC (see 3.24), although one other variant did have a better win-loss rating. Despite the noise in the data, this result appears significant enough to show some improvement. The generalization of this to other domains is always debatable, which is why standout results in these experiments are highly desired. The variant that showed this improvement in median MRE used only 10 boosting iterations each using a 75% subsample of the data.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC_Boost:10_Subsample:0.20	2258	129	0	129	40.13	26.37	0.54	0.903
LC_Boost:45_Subsample:0.30	2377	10	0	10	39.96	25.95	0.49	0.891
LC	2377	10	0	10	41.16	27.46	0.49	0.914
Cocomin_Boost:5_Subsample:0.40	2377	10	0	10	43.31	29.19	0.47	0.867
Cocomin	2376	10	1	9	44.19	30.35	0.46	0.880

Table 3.23: Selected Boosting by Subsampling Results on Coc81 Data)

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
Cocomin_Boost:5_Subsample:0.40	3314	132	0	132	61.83	24.32	0.59	0.821
Cocomin_Boost:10_Subsample:0.75	2970	99	0	99	50.86	18.98	0.50	0.655
LC_Boost:5_Subsample:0.95	3521	96	0	96	72.03	30.03	0.63	0.914
LC	3521	96	0	96	72.28	29.95	0.63	0.914
Cocomin	3530	87	0	87	67.42	28.17	0.62	0.884
LC_Boost:40_Subsample:0.50	3540	77	0	77	71.39	29.86	0.61	0.912

Table 3.24: Selected Boosting by Subsampling Results on Nasa93 Data)

3.4.2 Boosting by Oversampling

Another path to boosting is to oversample by including the most difficult instances in the calibration data multiple times. The code used in each of the oversampling experiments is available online at `http://unbox.org/wisp/var/dan/boosting/oversample/`.

1st Oversampling Experiment

The first boosting by oversampling algorithm that was tried worked by adding the two records that were predicting the worst back into the training set for the next boosting iteration. This choice was made to see if a small number of difficult instances were throwing the COCOMO algorithm off balance. If so, these would quickly get enough focus to classify them correctly.

This first boosting by oversampling boosted standard local calibration, and did not include and feature selectors.

This boosting algorithm's final prediction is the result of using the hypothesis from each boosting iteration to make a prediction, and then combining these estimates using a weighted average where the weight coefficient is the inverse of the MMRE of an evaluation of the oversampled dataset.

The results, shown in table 3.25, show that this boosting method's performance was inferior to standard LC. This could be because the presence of a small number of outliers became more dominant.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	49	8	0	8	55.38	26.82	0.53	0.847
BoostedLC_It:5_Worst+2	54	2	1	1	54.72	29.40	0.50	0.841
BoostedLC_It:10_Worst+2	54	0	3	-3	55.06	32.76	0.43	0.834
BoostedLC_It:15_Worst+2	51	0	6	-6	55.76	36.20	0.39	0.831

Table 3.25: 1st Boosting by Oversampling Experiment Results on All Data

2nd Oversampling Experiment

The 2nd boosting by oversampling tried boosting locally calibrated COCOMO by adding a percentage of the original calibration set back into the virtual set at each boosting iteration. These boosting algorithms are described by the number of boosting iterations and the percentage increased at each iteration.

This boosting algorithm's final prediction is the result of using the hypothesis from each boosting iteration to make a prediction, and then combining these estimates using a weighted average where the weight coefficient is the inverse of the MMRE of an evaluation of the oversampled dataset.

Table 3.26 illustrates one of the better performing boosting variations, which still did not do as well as LC, although its results were similar. Thus, this boosting added no improvement. The rest of the experiment's results are freely available at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	273	107	0	107	55.38	26.82	0.53	0.847
LC_Boost:5_Oversample:0.10	315	65	0	65	52.58	28.68	0.51	0.843

Table 3.26: Selected Boosting by Oversampling 2nd Results on All Data)

3rd Oversampling Experiment

The next boosting algorithm worked the same as the one from the 2nd experiment, except it included the use of Cocomin and a boosted Cocomin.

Once again, the results, illustrated in table 3.27, made no improvement over simpler methods. The rest of the experiment's results are freely available at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
Cocomin_Boost:15_Oversample:0.30	1322	65	0	65	52.62	27.53	0.53	0.823
LC	1325	62	0	62	55.38	26.82	0.53	0.848
Cocomin	1337	50	0	50	54.51	26.73	0.52	0.820
LC_Boost:5_Oversample:0.30	1342	45	0	45	53.93	27.41	0.53	0.846

Table 3.27: Selected Boosting by Oversampling 3rd Results on All Data)

4th Oversampling Experiment

The final boosting by oversampling algorithm tried was the same as the one in the 3rd experiment, except each boosting variant was evaluated on its estimation ability of the original dataset, instead of the virtual (oversampled) dataset.

The results, illustrated in table 3.28, produced roughly equivalent results as simpler methods. Thus, the addition of the boosting was superfluous. The rest of the experiment's results are freely available at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
Cocomin_Boost:70_Oversample:0.35	6245	614	0	614	52.55	27.45	0.53	0.828
Cocomin_Boost:25_Oversample:0.40	6278	581	0	581	53.31	26.76	0.52	0.830
LC	6331	528	0	528	55.38	26.82	0.53	0.847
Cocomin	6354	505	0	505	54.51	26.73	0.52	0.822
LC_Boost:5_Oversample:0.45	6377	482	0	482	54.88	27.42	0.52	0.846
LC_Boost:5_Oversample:0.45	6377	482	0	482	54.88	27.42	0.52	0.846

Table 3.28: Selected Boosting by Oversampling 4th Results on All Data)

3.4.3 Distributive Boosting with Adaboost

A final boosting technique is to distributively sample the instances, such as with the popular Adaboost boosting algorithm. In this method, weights are assigned to each instance, and virtual training sets are created by probabilistically sampling from this set. Adaboost is described in detail in section 2.8.3 and illustrated in figure 2.8.

The code used in the Adaboost experiments is available online at `http://unbox.org/wisp/var/dan/boosting/adaboost/`.

1st Adaboost Experiment

The first "Adaboost" experiment used wasn't technically Adaboost because a slight change was made that was believed to lead to better performance than the original Adaboost algorithm. This difference occurs in the weights assigned to each calibration instance that affect its choice in the future distribution of calibration data for boosting iterations. The original Adaboost algorithm was based on a classification system, and casting Adaboost to regression problems involved using a threshold. If the MMRE of a given instance exceeded the threshold then it was considered misclassified, else it was successfully classified. This way updating the weights occurs the same as with traditional Adaboost.

In this first experiment, the weights were instead changed to equal the MRE of the instance during the last evaluation with the training data. This allows the ”misclassified” instances to be selected proportionally to the error of their classification.

The boosting algorithms used in this experiment are described in the tables with the number of boosting iterations performed. The estimates of each iteration were combined using a weighted average with the inverse MMRE as the weighting coefficient.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	702	77	0	77	55.38	26.82	0.53	0.844
Cocomin	733	46	0	46	54.51	26.73	0.52	0.820
Cocomin_DistrBoost_v1:It:15	759	20	0	20	52.47	30.38	0.48	0.814
LC_DistrBoost_v1:It:5	775	0	4	-4	54.71	31.67	0.48	0.831

Table 3.29: Selected Adaboost 1st Results on All Data)

The results from this experiment, illustrated in tables 3.29, 3.30, and 3.31, found that the boosting did not create an improvement, and sometimes deteriorated performance. Simply put, the extra CPU clock cycles were a waste. The rest of the experiment’s results are freely available at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	287	0	0	0	41.16	27.46	0.49	0.915
Cocomin	287	0	0	0	44.19	30.35	0.46	0.880
Cocomin_DistrBoost_v1:It:85	287	0	0	0	41.87	28.72	0.50	0.865
LC_DistrBoost_v1:It:55	287	0	0	0	39.82	26.13	0.53	0.916

Table 3.30: Selected Adaboost 1st Results on Coc81 Data)

2nd Adaboost Experiment

The second Adaboost experiment used the proper version of Adaboost instead of the version in the first experiment that was modified with hopes of greater performance. Unfortunately, this version of Adaboost didn’t help any either, as can be seen in tables 3.32, 3.33, and 3.34. This was

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
LC	415	77	0	77	63.67	26.44	0.56	0.802
Cocomin	446	46	0	46	60.53	24.62	0.55	0.785
Cocomin_DistrBoost_v1:It:15	472	20	0	20	58.79	31.19	0.47	0.782
LC_DistrBoost_v1:It:5	488	0	4	-4	63.56	33.30	0.45	0.785

Table 3.31: Selected Adaboost 1st Results on Nasa93 Data)

surprising considering the impressive results of Adaboost in the literature. One possible cause is that Adaboost is supposed to be used on a *weak*, or very simple, learner. The learners used in this study may have been too complex. It would be interesting to see Adaboost's performance using only the relationship between KLOC and the effort, ignoring all the cost drivers.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
Cocomin	167	4	0	4	54.51	26.73	0.52	0.821
Cocomin_DistrBoost_Adaboost:It:10	167	4	0	4	58.43	29.73	0.48	0.818
Cocomin_DistrBoost_Adaboost:It:40	167	4	0	4	60.47	30.70	0.47	0.820
Cocomin_DistrBoost_Adaboost:It:20	169	2	0	2	59.78	31.10	0.47	0.816
Cocomin_DistrBoost_Adaboost:It:30	169	2	0	2	59.64	30.39	0.49	0.818
LC	171	0	0	0	55.38	26.82	0.53	0.848
LC_DistrBoost_Adaboost:It:10	168	0	3	-3	62.32	28.69	0.51	0.852
LC_DistrBoost_Adaboost:It:20	168	0	3	-3	64.55	29.31	0.49	0.852
LC_DistrBoost_Adaboost:It:30	166	0	5	-5	66.29	29.38	0.49	0.845
LC_DistrBoost_Adaboost:It:40	166	0	5	-5	66.45	29.00	0.49	0.847

Table 3.32: Results from 2nd Adaboost Experiment on All Data

3.5 Summary of Experimental Results

The most dramatic result from these experiments is that standard locally calibrated COCOMO never lost one of its nonparametric Mann Whitney Wilcoxon rank-sum tests at a 95% confidence level. Some of the new methods tried did just as well, and possibly better if secondary characteristics such as median MRE are considered.

The experiments on feature subset selection algorithms found FSS to be useful on the NASA93 data and its subsets. An exhaustive search of the attribute space, COCOMOST, scored about the same as both a more intermediate search strategy, dBFS, and a very simple near-linear time search,

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
Cocomin	59	4	0	4	44.19	30.35	0.46	0.881
Cocomin_DistrBoost_Adaboost:It:10	59	4	0	4	48.76	34.14	0.42	0.879
Cocomin_DistrBoost_Adaboost:It:40	59	4	0	4	52.50	36.07	0.40	0.864
Cocomin_DistrBoost_Adaboost:It:20	61	2	0	2	51.26	36.60	0.38	0.874
Cocomin_DistrBoost_Adaboost:It:30	61	2	0	2	51.85	35.66	0.42	0.874
LC	63	0	0	0	41.16	27.46	0.49	0.915
LC_DistrBoost_Adaboost:It:10	60	0	3	-3	47.00	33.16	0.44	0.920
LC_DistrBoost_Adaboost:It:20	60	0	3	-3	49.03	33.82	0.42	0.918
LC_DistrBoost_Adaboost:It:30	58	0	5	-5	50.15	33.72	0.43	0.909
LC_DistrBoost_Adaboost:It:40	58	0	5	-5	51.12	33.85	0.43	0.915

Table 3.33: Results from 2nd Adaboost Experiment on Coc81 Data

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
Cocomin	108	0	0	0	60.53	24.62	0.55	0.785
Cocomin_DistrBoost_Adaboost:It:10	108	0	0	0	64.07	27.16	0.52	0.782
Cocomin_DistrBoost_Adaboost:It:20	108	0	0	0	64.75	27.88	0.51	0.781
Cocomin_DistrBoost_Adaboost:It:30	108	0	0	0	64.18	27.32	0.53	0.786
Cocomin_DistrBoost_Adaboost:It:40	108	0	0	0	65.11	27.56	0.51	0.794
LC	108	0	0	0	63.67	26.44	0.56	0.809
LC_DistrBoost_Adaboost:It:10	108	0	0	0	71.26	26.09	0.55	0.812
LC_DistrBoost_Adaboost:It:20	108	0	0	0	73.60	26.67	0.54	0.814
LC_DistrBoost_Adaboost:It:30	108	0	0	0	75.70	26.84	0.52	0.807
LC_DistrBoost_Adaboost:It:40	108	0	0	0	75.39	26.17	0.53	0.806

Table 3.34: Results from 2nd Adaboost Experiment on Nasa93 Data

COCOMIN. Therefore, due to the comparable performance, the fastest approach, COCOMIN, is recommended.

The experiments into the use of bagging of the COCOMO based methods did not produce noticeable gains.

Most of the experiments run on the use of boosting of COCOMO based methods did not produce noticeable gains. However, on the Nasa93 data, a boosting by subsampling variant which used 10 boosting iterations of 75% subsamples of the COCOMIN learner reported a median MRE about 11% lower than standard LC (see 3.24). This result suggests subsampling may be useful in certain cases. Surprisingly, the use of Adaboost did not make an improvement.

Chapter 4

The Methodology of 2CEE

4.1 Introduction

This section describes the theory motivating a new software effort estimation methodology called *2CEE*, or *21st Century Effort Estimation Methodology*. This methodology was implemented in an industrial application for NASA's Jet Propulsion Laboratory in Pasadena, California. All of the details related to this implementation are in the chapter titled *Practice*.

2CEE is motivated by the current literature on software effort estimation, see Chapter 2, as well as the results of the laboratory studies described in chapter 3. These studies show which model-based methods have been found to be useful or not. Based on these results, the direction of the thesis research transitioned into the industrial application of these methods. The collaborative research with Tim Menzies and Omid Jalali of West Virginia University, and Jairus Hihn and Karen Lum of the Jet Propulsion Laboratory greatly inspired the new methodology described in this chapter.

4.2 Combining Aspects of Experts and Models

The proposed methodology makes progress in adding characteristics of expert-judgment to software effort estimation models in two primary ways. First, the cost analyst, or expert, should be more involved in the model's estimate. The model's operation shouldn't be viewed as a ”black box” from the expert, allowing them to be ignorant of the model's decision making structure. It is important for the expert to understand both how the model works and can see and guide its operation to avoid errors of assumption and scope. In addition, this provides more business level insight, and benefits from the ”sanity check” of the analyst if the model isn’t operating correctly. Second, the proposed methodology attempts to implement as many of Jorgensen’s expert estimation best-practices [49], shown in section 2.4, as possible. If these best practices can be achieved then perhaps the distinction between expert and model systems is suspect for review. More information on the best practices implemented can be found in section 5.1. Although not all of these practices were implemented, future work could implement them.

4.3 Model Calibration Approach in 2CEE

The initial motivation for 2CEE was to produce an effort estimation toolkit with an intuitive user interface for the Jet Propulsion Laboratory that contains many of the new estimation techniques being developed at West Virginia University. Coincidentally this overlaps with the first way this research attempts to combine model and expert judgment as described above. The proposed methodology provides a framework that allows the cost analyst to try different estimation models, experiment with calibration options, and visualize aspects of the historical data.

Due to the literature on feature selection, section 2.6, and record selection techniques, section 2.7, each of these data pruning techniques are standard options in the proposed methodology. Manual options for both feature and record selection should be available to easily remove outliers identified by the cost analyst. Options to facilitate the reduction of the historical data to a stratified subset should be included. For example, the user may want to use only flight records from a NASA dataset including both flight and ground projects. This should only take a few button clicks provided a file with descriptors of each historical record exists.

In addition, automated methods for feature and record selection should be included. The results of section 3.2 show that a simple, nearly linear time feature selector such as Cocomin should be sufficient, at least for COCOMO data. The automated nearest neighbor row selection algorithm LOCOMO, designed by Jalali [46] and introduced by our previous study [95], is a promising preprocessor for regression estimators like COCOMO. LOCOMO is described in section 2.7.2.

Although the focus of this research has been applied to algorithms derived from COCOMO, the proposed methodology is meant to include multiple models and this is an ideal candidate for future work. In addition, the suggested 2CEE approach is a framework that easily allows the addition of more models over time.

Finally, designing model calibration to incorporate uncertainty is a key requirement to successfully managing it as described in the next subsection.

4.4 Managing Uncertainty

The problem of uncertainty in software effort estimation is detailed in section 2.5.4. The proposed methodology, 2CEE, should represent the uncertainty of the estimate. Thus, the final estimate should not be a point estimate but instead a probabilistic estimate or range of estimates. Sorting a range of estimates into an estimate curve is an intuitive visualization of uncertainty. Unfortunately, visualization isn't the problem. Rather, the real question is, ”How much uncertainty is there in my estimate?”

A natural way to determine the true uncertainty of an estimate is to represent all of its potential sources. These are the same as the sources of error in an estimate. Kitchenham and Linkman have described four sources of estimate uncertainty [68].

- *Measurement error* is error from accuracy limitations in the input variables of the model.
- *Model error* is error due to the model's abstractions from reality.
- *Assumption error* is error from making incorrect assumptions about a model's input parameters.
- *Scope error* is error from estimating outside the model's domain.

4.4.1 Addressing Measurement and Assumption Error

Measurement and assumption error can be categorized together because they may be accommodated in the same way: by representing the uncertainty in the input parameters. There are many ways to represent the uncertainty of input parameters, as can be seen in section 2.5.4.

In models such as COCOMO, there are two types of input parameters to consider in this way. The first is the new project data. The second is the historical dataset; in COCOMO this results in the coefficients, *a* and *b*, which define the relationship between size and effort.

A simple and intuitive representation for the new project parameters is to use triangular distributions, i.e. a minimum, mode, and maximum value for each feature. This can be sampled using a Monte Carlo simulation to generate a range of project instances to estimate in the model. By representing the new project data in this way, estimates may be made very early in the life-cycle despite large uncertainties about the project. As time passes, information on the new project becomes more clear and the estimate may be refined. In this way, 2CEE is useful throughout the life cycle of software development.

Considering a COCOMO based model, at first one may consider a similar sampling approach for the COCOMO coefficients instead of representing the uncertainty directly at the data used to create the coefficients. This premise is flawed because *a* and *b* are inversely correlated. Instead, sampling may be done from *a*,*b* pairs. However, the only correct way to create these pairs is to represent the uncertainty of the historical data, and calibrate the model with instances sampled from this uncertain historical dataset. Thus, sampling the coefficients properly does not simplify the problem.

There are many ways (see section 2.5.4) to represent the uncertainty of a historical dataset *if it is collected to show it*. Unfortunately, this research had access only to historical data collected using point values. It is unlikely that the collected data is perfectly correct. For instance, Kemerer's work [59] found function point collection to be at least 12% inaccurate. What can be done to represent to the uncertainty in a historical dataset collected without it? Two potential solutions include using bootstrapping and injecting variance into the dataset.

Bootstrapping is a popular statistical operation used in many application that works by random sampling with replacement lots of times. It is described in more detail in section 2.5.3. The benefit of using bootstrapping to show the uncertainty from the historical data is that it does not make an assumption about how much ambiguity exists. Rather, the data determines the uncertainty. The drawback of using it in this case is that each bootstrap sample effectively uses a subset of the available calibration data. If a lot of data exists this is not much of a problem. However, in a domain like software effort estimation where very small datasets are the norm, this could result in using too little data which may lead to more inaccurate estimates.

Instead of bootstrapping, a method that injects variance into the dataset could use all of the available records for every calibration. However, the drawback introduced is that the user must make an assumption about the magnitude of uncertainty in the dataset. In this paper, particularly in the results tables of Chapter 5, this method is known as the *Fuzzy* method.

Implementing this ”Fuzzy” variance injection technique includes the following steps:

1. Define the assumed level of uncertainty in the historical data.

2. Define the distribution to use, such as Gaussian or triangular.

3. Many times create a mutated dataset by sampling from the original with the given distribution and variance.

4. Calibrate the model using this mutated dataset and make an estimate (or with COCOMO save the coefficient pair).

4.4.2 Addressing Model Error

Kitchenham proposes to assess model error by a measure such as mean magnitude of relative error [68]. Due to the non-Gaussian distribution of errors seen in effort estimation [95], this thesis instead endorses using the median MRE from a cross validation experiment. This value can then be used to stretch the sorted range of estimates. For example, given a 30% median MRE, the 0 percentile estimate will become 30% lower, the 100 percentile estimate will become 30% higher, and the 50 percentile estimate will stay the same. The other estimates will be linearly affected somewhere in between. If this technique is used don’t be surprised if the estimate curve looks more like a line than an ’S’; this is a natural consequence of the transformation.

4.4.3 Addressing Scope Error

Scope error is the most difficult source of uncertainty to address because an estimation model cannot perform confidently outside of its domain. Kitchenham warns, ”If your estimation models or methods are completely out of scope, you cannot produce a meaningful effort estimate [68].”

With a COCOMO based estimation methodology, scope error boils down to estimating a project that is very dissimilar from the historical experience. In this case, the standard practice is to use Boehm's default calibration of COCOMO. In this case, the best solution available is to use the default COCOMO's error seen in similar areas of the industry as the basis for the level of uncertainty.

For more similar projects in which local calibration is justified, the use of a row selection algorithm such as LOCOMO will help mitigate the effect of scope error by calibrating on the records most similar to the new project.

4.5 Evaluation Tools

A big benefit of using a model based approach is that the historical data available can be used to evaluate the effectiveness of the model, typically using cross-validation (see section 2.5.3). The estimation framework should have tools available to easily run this cross validation and analyze the results. It is especially important for the cost analyst to be able to easily run this analysis in an environment where they can tweak and customize the algorithms used in the model. This type of analysis is important to understand the accuracy of a single learner.

Another useful evaluation tool for the proposed framework is one that allows a batch experiment to be run involving the cross-validation results from multiple algorithms on multiple datasets. Ideally analysis tools should also be provided. These tools will allow the cost analyst to better understand their organizational domain's relationship to the existing algorithms, and to know which ones tend to work best in their organization.

Chapter 5

Industrial Studies

This chapter describes the implementation results of the *21st Century Effort Estimation Methodology* described in the previous chapter. The resulting tool,*2CEE*, is currently available from the Jet Propulsion Laboratory. To obtain a copy, send an email to `softwarerelease@jpl.nasa.gov` with your name, citizenship, affiliation, and the name of the software that you are requesting, 2CEE.

5.1 Implemented Expert Estimation Best Practices

The following seven expert estimation best practices from the original list of Jorgensen's 12 [49] were implemented in 2CEE:

1. evaluate estimation accuracy, but avoid high evaluation pressure
2. ask the estimators to justify and criticize their estimates
3. avoid irrelevant and unreliable estimation information
4. use documented data from previous development tasks
5. assess the uncertainty of the estimate
6. provide feedback on estimation accuracy
7. find estimation experts with relevant domain background

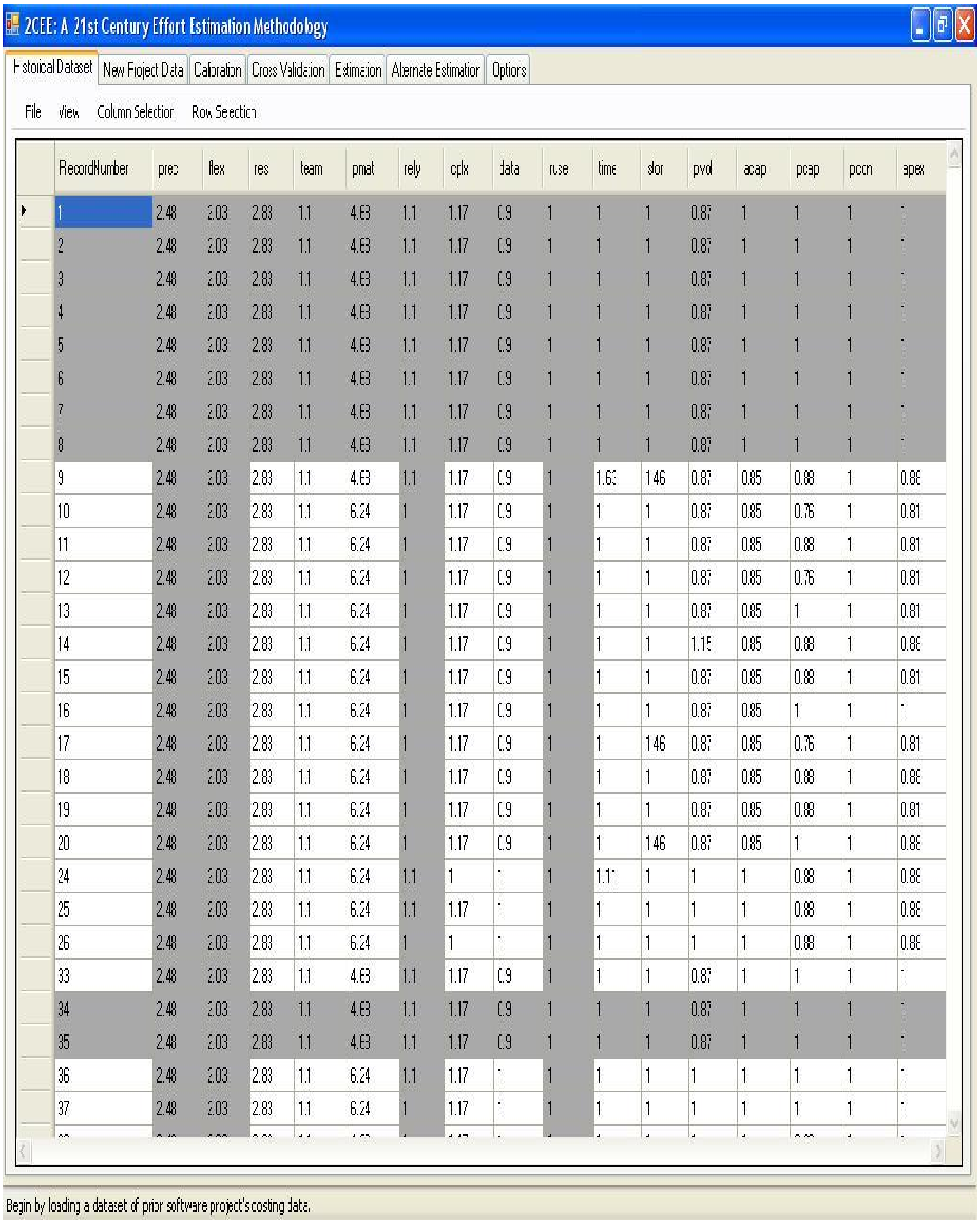
The first of these six practices was handled by using a cross validation experiment of the historical data with the model in question to generate a range of relative error values. This avoided high evaluation pressure by taking the human out of the loop. The second practice was handled both inherently by the use of a model calibrated on local data, the use of the well known and respected COCOMO model, as well as the cross validation previously mentioned. The third practice was accomplished by the use of feature selection algorithms, such as COCOMIN, and record selection algorithms, such as LOCOMO. These algorithms remove irrelevant and unreliable data from use. The fourth practice was accomplished by use of the historical dataset used to calibrate the model. The fifth practice was accomplished by representing the uncertainty in the project ranges with triangular distributions, the uncertainty in the historical data either through bootstrapping or variance injection, and the uncertainty from model error by stretching the estimate by the median MRE from the cross validation, and finally providing a range of estimates to the analyst. The sixth practice may be accomplished by making an estimate for a project in which you don't know the actual effort, and then having the actual effort revealed to you. This can be done much easier and faster with the 2CEE models than with expert-judgment. The seventh practice, "find estimation experts with relevant domain background", is accomplished by calibrating good models with relevant project data. In this way, the model is thought of as a digital expert.

Additional best practices may be implemented in future work as discussed in section 6.1.2.

5.2 An Industrial Implementation of the 2CEE Methodology

The 2CEE tool was built using a combination of Visual Basic and Visual C++. It has a graphical user interface with several tabs relating to various stages of estimation. There is a tab for managing historical data, one for new project data, one for visualizing calibration results, one for cross validation, two for estimation (one is simply meant for plug-ins), and a final tab for options. Logs of estimates may be saved which include all of the relevant information used to make the estimate.

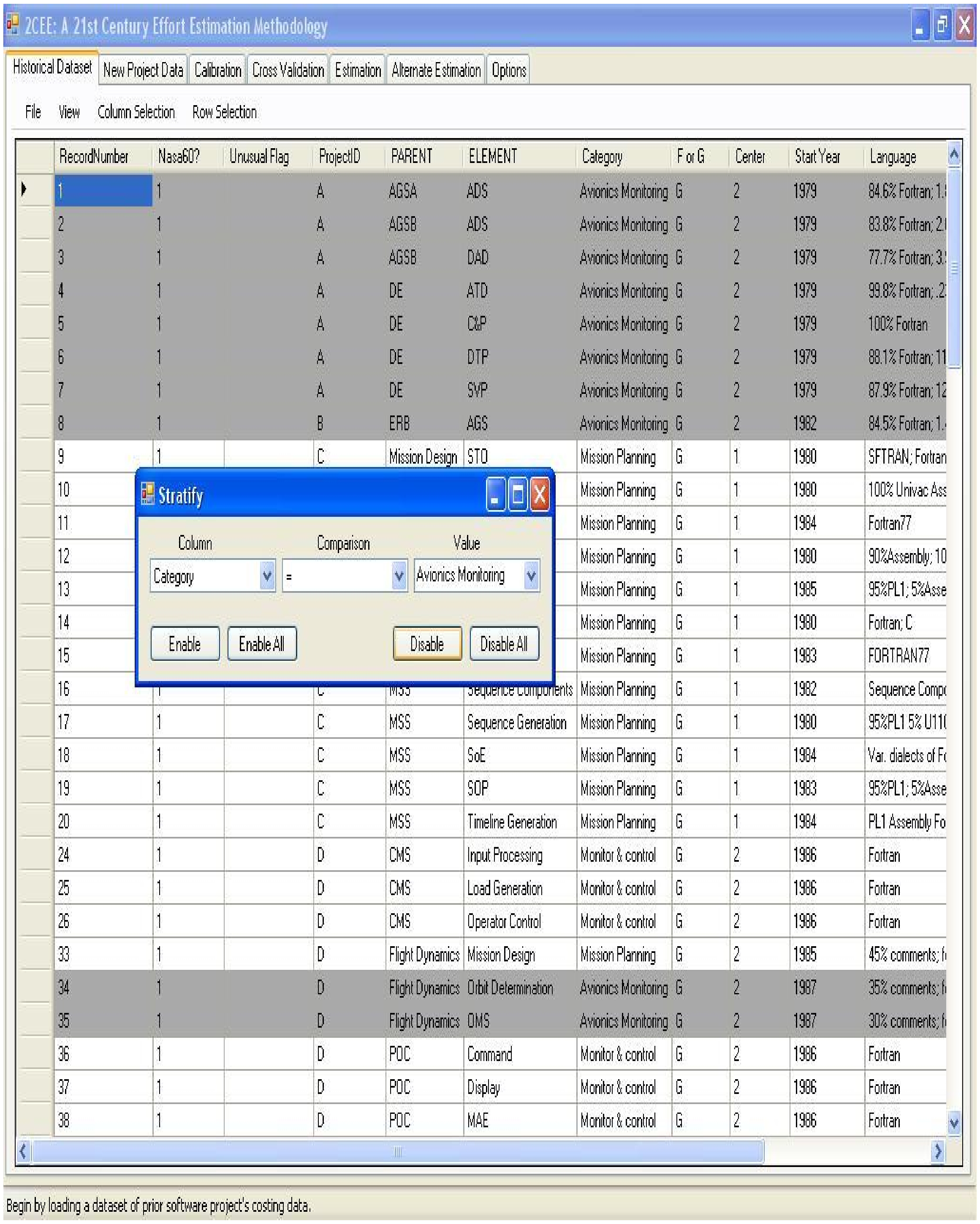
The 2CEE tool was built as described in chapter 4, although there are aspects weren't able to be implemented to their full potential due to time constraints. For example, 2CEE should have more learners than variants of COCOMIN using either row or feature selection. In addition, the batch experimenter mode that was built does not include does not include evaluation functions. Finally, more work can be done towards using 2CEE to combine expert and model practices. Nevertheless, 2CEE has a great deal of functionality.



RecordNumber	prec	flex	resl	team	pmat	rely	cplx	data	ruse	time	stor	pvol	acap	pcap	pcon	apex
1	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
2	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
3	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
4	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
5	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
6	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
7	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
8	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
9	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1.63	1.46	0.87	0.85	0.88	1	0.88
10	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	0.87	0.85	0.76	1	0.81
11	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	0.87	0.85	0.88	1	0.81
12	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	0.87	0.85	0.76	1	0.81
13	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	0.87	0.85	1	1	0.81
14	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	1.15	0.85	0.88	1	0.88
15	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	0.87	0.85	0.88	1	0.81
16	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	0.87	0.85	1	1	1
17	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1.46	0.87	0.85	0.76	1	0.81
18	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	0.87	0.85	0.88	1	0.88
19	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1	0.87	0.85	0.88	1	0.81
20	2.48	2.03	2.83	1.1	6.24	1	1.17	0.9	1	1	1.46	0.87	0.85	1	1	0.88
24	2.48	2.03	2.83	1.1	6.24	1.1	1	1	1	1.11	1	1	1	0.88	1	0.88
25	2.48	2.03	2.83	1.1	6.24	1.1	1.17	1	1	1	1	1	1	0.88	1	0.88
26	2.48	2.03	2.83	1.1	6.24	1	1	1	1	1	1	1	1	0.88	1	0.88
33	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
34	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
35	2.48	2.03	2.83	1.1	4.68	1.1	1.17	0.9	1	1	1	0.87	1	1	1	1
36	2.48	2.03	2.83	1.1	6.24	1.1	1.17	1	1	1	1	1	1	1	1	1
37	2.48	2.03	2.83	1.1	6.24	1	1.17	1	1	1	1	1	1	1	1	1

Begin by loading a dataset of prior software project's costing data.

Figure 5.1: This figure shows the management of historical data in 2CEE. Each row is a software project and each column is a feature of these projects. The gray rows and columns show that the representative records and features have been disabled. The records may be sorted by the columns.

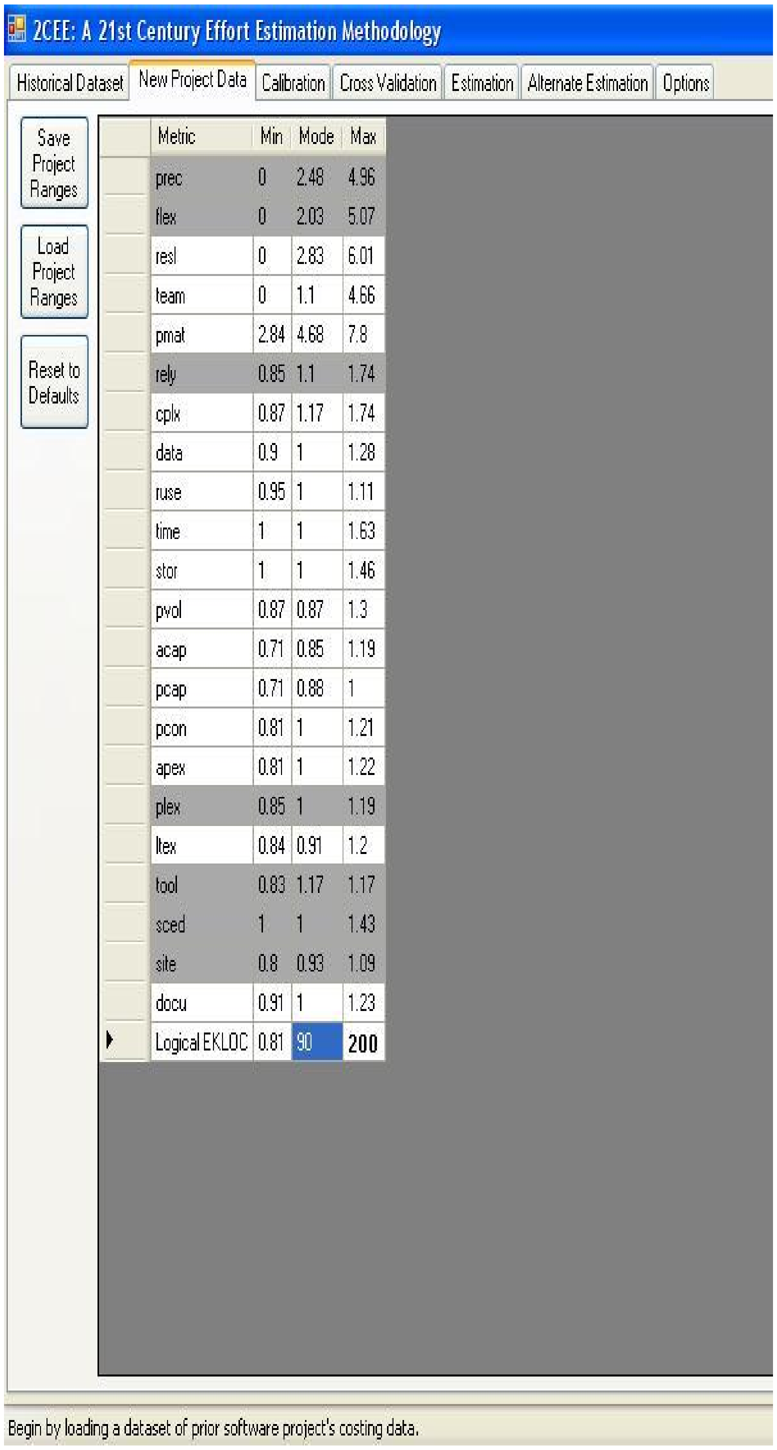


	RecordNumber	Nasa60?	Unusual Flag	ProjectID	PARENT	ELEMENT	Category	F or G	Center	Start Year	Language
▸	1	1		A	AGSA	ADS	Avionics Monitoring	G	2	1979	84.6% Fortran; 1.
	2	1		A	AGSB	ADS	Avionics Monitoring	G	2	1979	83.8% Fortran; 2.
	3	1		A	AGSB	DAD	Avionics Monitoring	G	2	1979	77.7% Fortran; 3.
	4	1		A	DE	ATD	Avionics Monitoring	G	2	1979	99.8% Fortran; .2
	5	1		A	DE	C&P	Avionics Monitoring	G	2	1979	100% Fortran
	6	1		A	DE	DTP	Avionics Monitoring	G	2	1979	88.1% Fortran; 11
	7	1		A	DE	SYP	Avionics Monitoring	G	2	1979	87.9% Fortran; 12
	8	1		B	ERB	AGS	Avionics Monitoring	G	2	1982	84.5% Fortran; 1.
	9	1		C	Mission Design	STO	Mission Planning	G	1	1980	SFTRAN; Fortran
	10						Mission Planning	G	1	1980	100% Univac Ass
	11						Mission Planning	G	1	1984	Fortran77
	12						Mission Planning	G	1	1980	90%Assembly; 10
	13						Mission Planning	G	1	1985	95%PL1; 5%Asse
	14						Mission Planning	G	1	1980	Fortran; C
	15						Mission Planning	G	1	1983	FORTRAN77
	16	1		C	MSS	Sequence Components	Mission Planning	G	1	1982	Sequence Compo
	17	1		C	MSS	Sequence Generation	Mission Planning	G	1	1980	95%PL1 5% U11
	18	1		C	MSS	SoE	Mission Planning	G	1	1984	Var. dialects of F
	19	1		C	MSS	SOP	Mission Planning	G	1	1983	95%PL1; 5%Asse
	20	1		C	MSS	Timeline Generation	Mission Planning	G	1	1984	PL1 Assembly Fo
	24	1		D	CMS	Input Processing	Monitor & control	G	2	1986	Fortran
	25	1		D	CMS	Load Generation	Monitor & control	G	2	1986	Fortran
	26	1		D	CMS	Operator Control	Monitor & control	G	2	1986	Fortran
	33	1		D	Flight Dynamics	Mission Design	Mission Planning	G	2	1985	45% comments; f
	34	1		D	Flight Dynamics	Orbit Determination	Avionics Monitoring	G	2	1987	35% comments; f
	35	1		D	Flight Dynamics	OMS	Avionics Monitoring	G	2	1987	30% comments; f
	36	1		D	POC	Command	Monitor & control	G	2	1986	Fortran
	37	1		D	POC	Display	Monitor & control	G	2	1986	Fortran
	38	1		D	POC	MAE	Monitor & control	G	2	1986	Fortran

Column	Comparison	Value
Category	=	Avionics Monitoring

Begin by loading a dataset of prior software project's costing data.

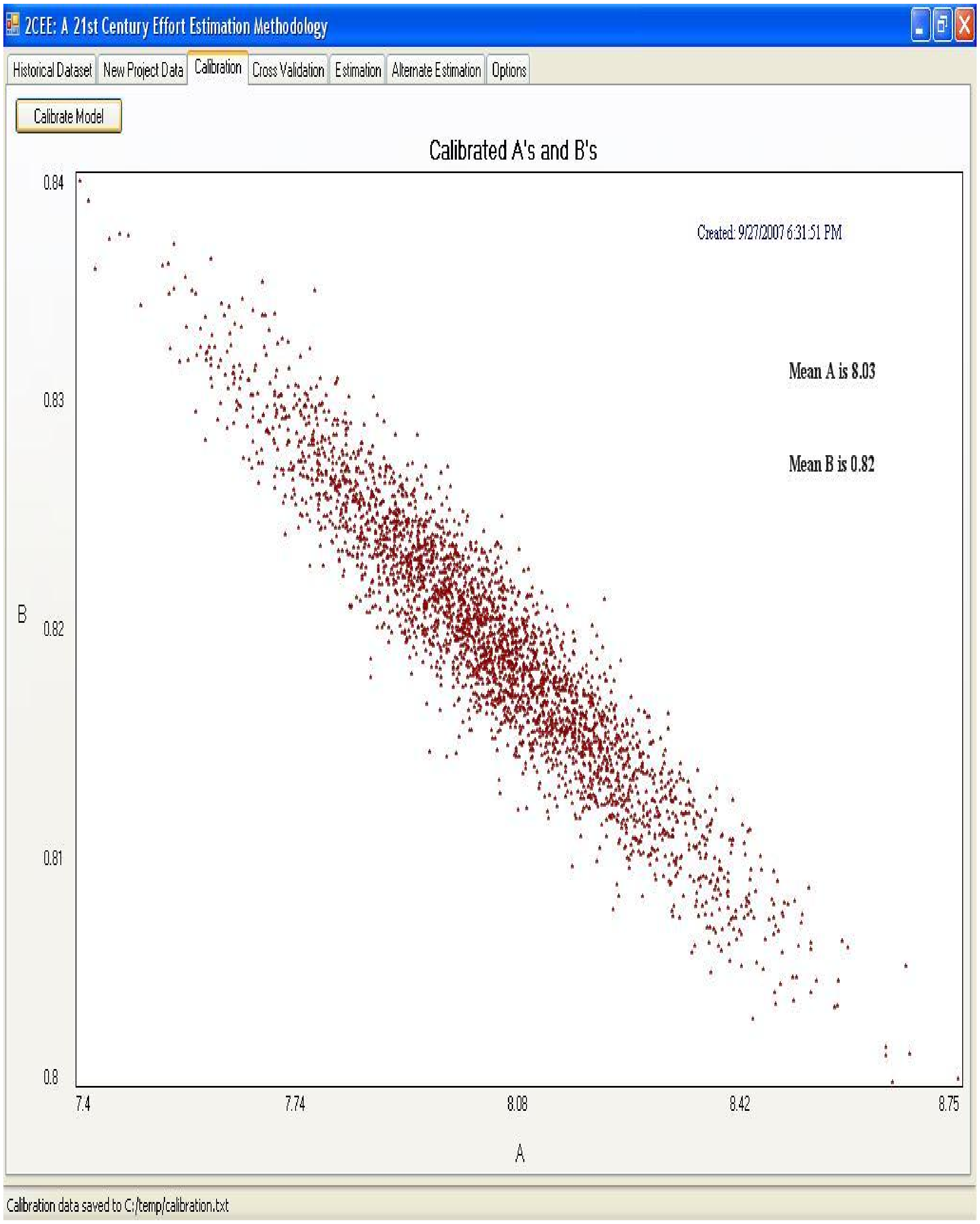
Figure 5.2: This figure shows use of the stratification tool in 2CEE. Given a file of descriptors, the tool can enable or disable records based on their descriptor values.



Metric	Min	Mode	Max
prec	0	2.48	4.96
flex	0	2.03	5.07
resl	0	2.83	6.01
team	0	1.1	4.66
pmat	2.84	4.68	7.8
rely	0.85	1.1	1.74
cplx	0.87	1.17	1.74
data	0.9	1	1.28
ruse	0.95	1	1.11
time	1	1	1.63
stor	1	1	1.46
pvol	0.87	0.87	1.3
acap	0.71	0.85	1.19
pcap	0.71	0.88	1
pcon	0.81	1	1.21
apex	0.81	1	1.22
plex	0.85	1	1.19
ltex	0.84	0.91	1.2
tool	0.83	1.17	1.17
sced	1	1	1.43
site	0.8	0.93	1.09
docu	0.91	1	1.23
Logical EKLOC	0.81	90	200

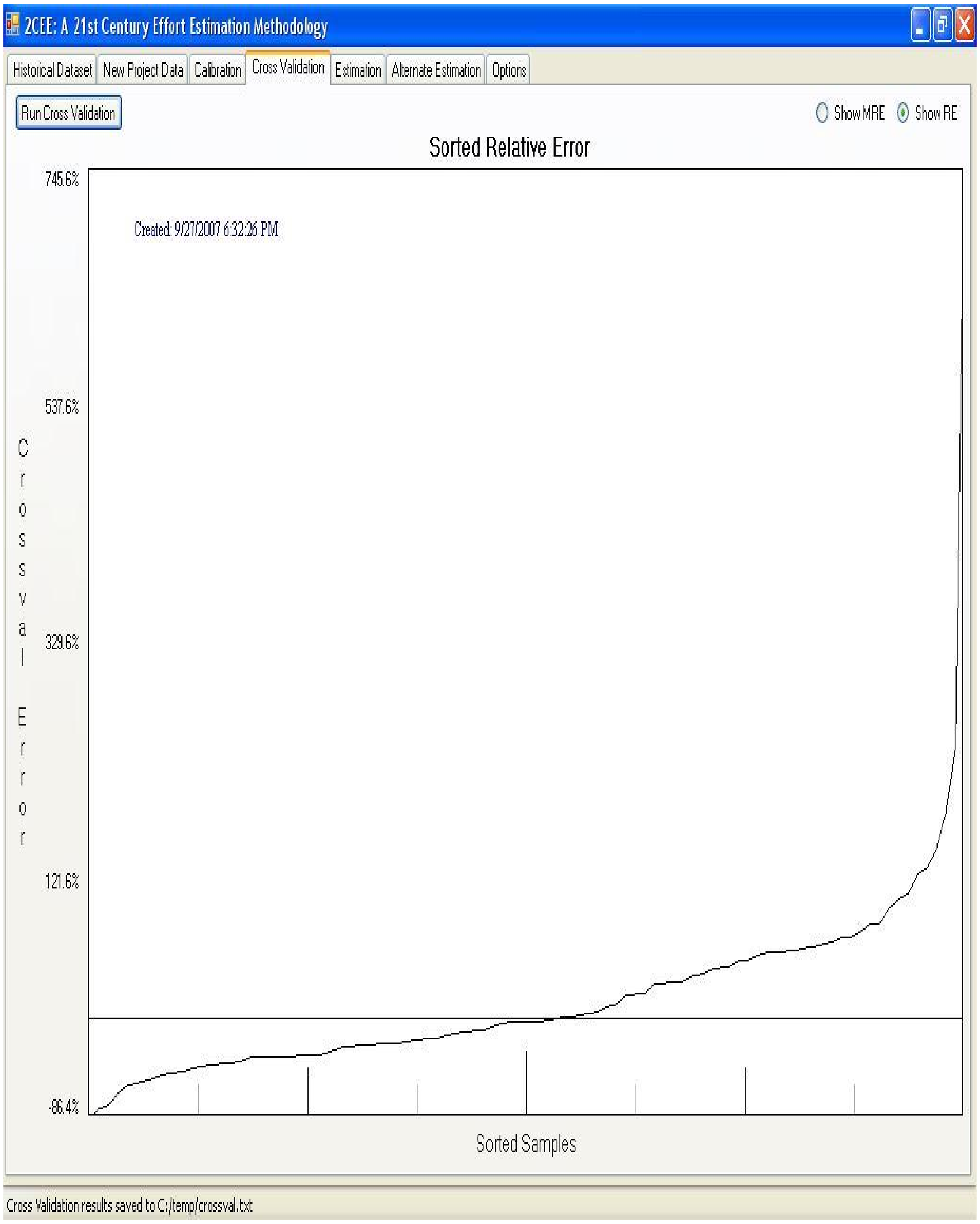
Begin by loading a dataset of prior software project's costing data.

Figure 5.3: This figure shows 2CEE describing a new project. Each cost driver is represented by a triangular distribution. Using ranges instead of a point value has numerous benefits, one of which is making estimation more useful both very early and throughout the development life-cycle. Grayed out features are disabled.



Calibration data saved to C:/temp/calibration.txt

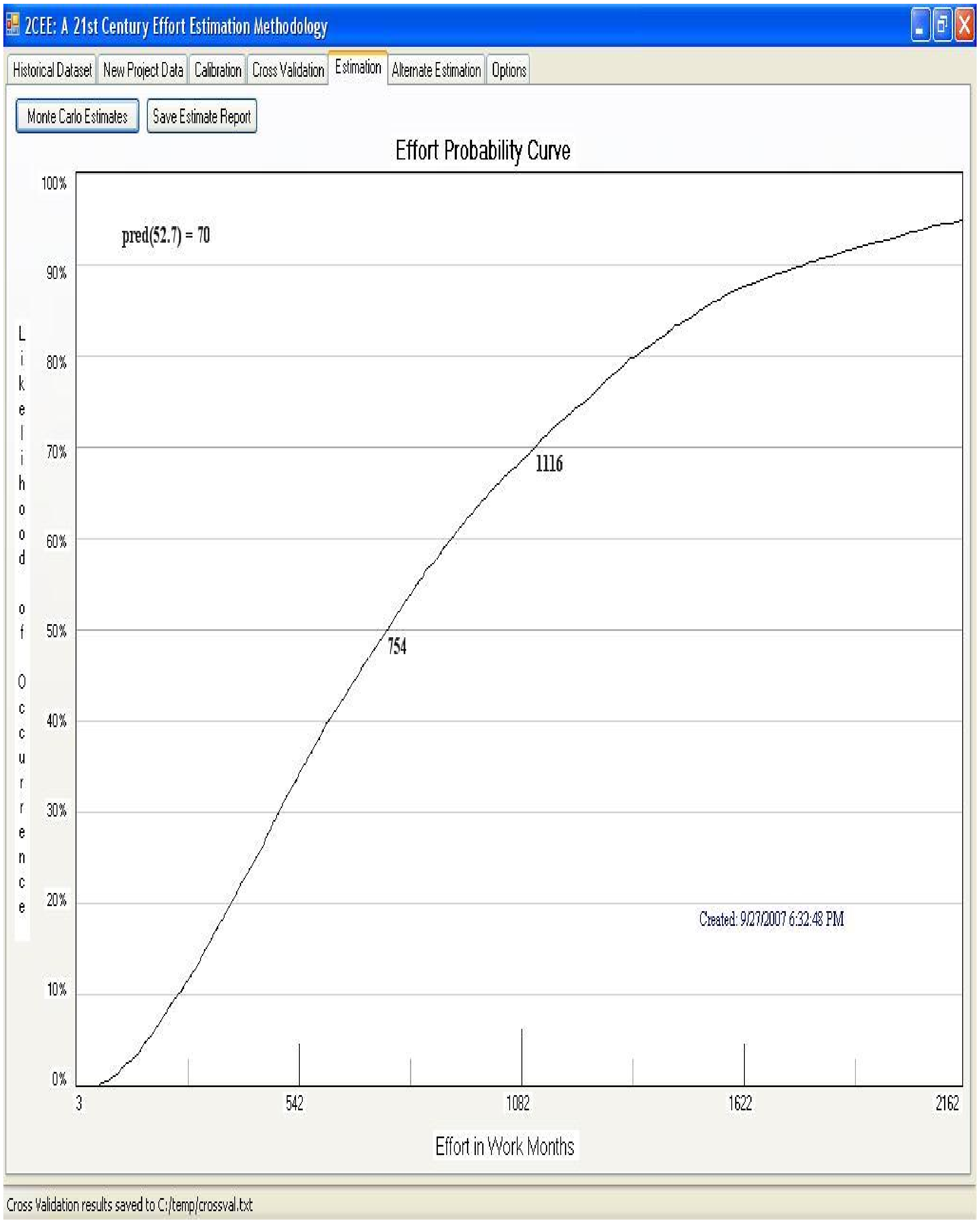
Figure 5.4: This figure shows the calibration panel. It shows the results of injecting a specified amount of uncertainty into the historical data and calibrating the model throughout this range. This produces a range of COCOMO coefficients which are visualized in this plot.



Created: 9/27/2007 6:32:26 PM

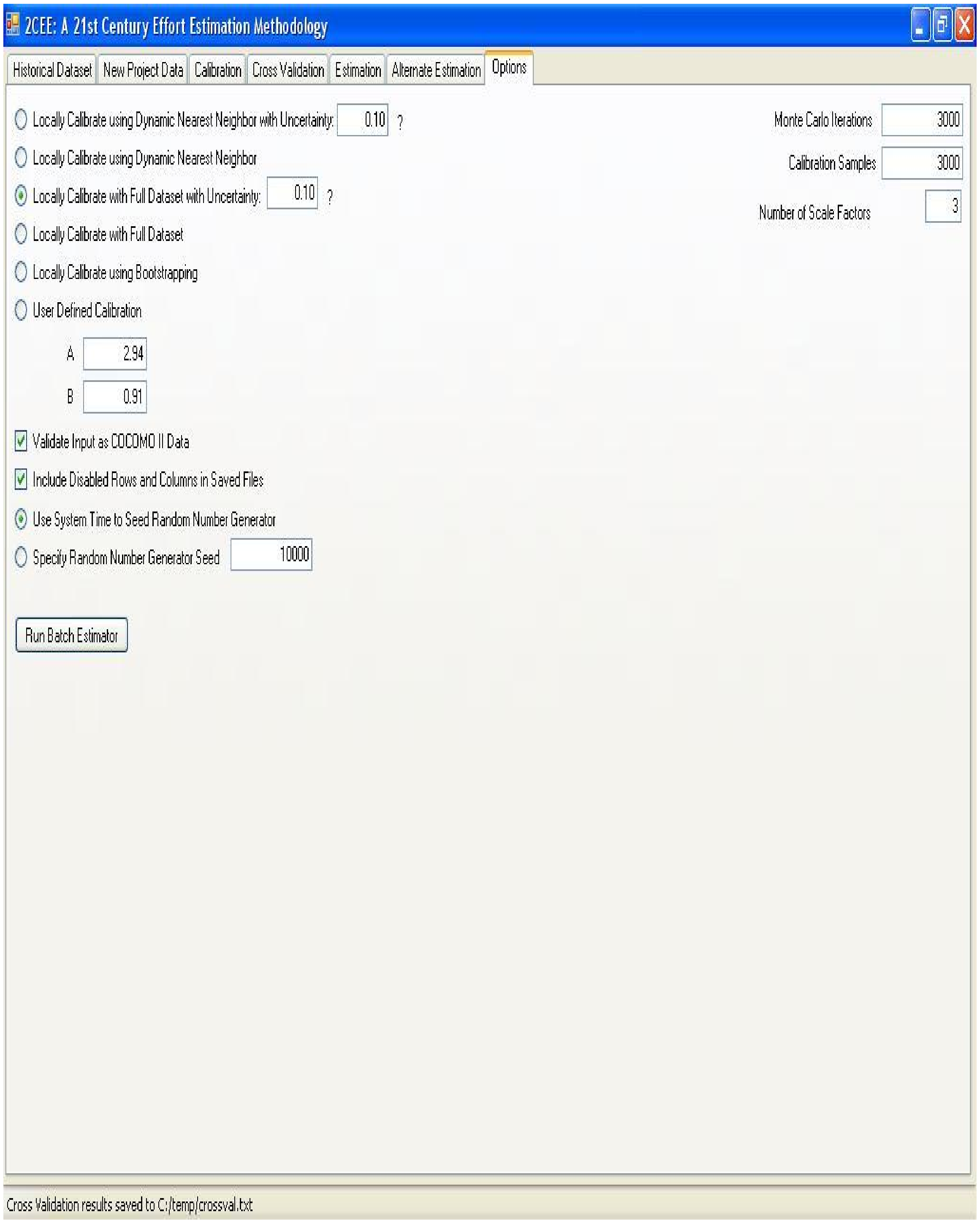
Cross Validation results saved to C:/temp/crossval.txt

Figure 5.5: This figure shows an example of leave-one-out cross validation in 2CEE. Clearly a small number of examples account for most of the error. The results may be viewed as either sorted relative error or sorted mean relative error.



Cross Validation results saved to C:/temp/crossval.txt

Figure 5.6: This figure shows an estimate in 2CEE. Instead of a single number, 2CEE provides a range of estimates with the likelihood that a given budget will finish the project. This allows for risk management.


2CEE: A 21st Century Effort Estimation Methodology

- ( ) Locally Calibrate using Dynamic Nearest Neighbor with Uncertainty: 0.10 ?
- ( ) Locally Calibrate using Dynamic Nearest Neighbor
- (✓) Locally Calibrate with Full Dataset with Uncertainty: 0.10 ?
- ( ) Locally Calibrate with Full Dataset
- ( ) Locally Calibrate using Bootstrapping
- ( ) User Defined Calibration

A 2.94

B 0.91

- [✓] Validate Input as COCOMO II Data
- [✓] Include Disabled Rows and Columns in Saved Files
- (✓) Use System Time to Seed Random Number Generator
- ( ) Specify Random Number Generator Seed 10000

Monte Carlo Iterations 3000

Calibration Samples 3000

Number of Scale Factors 3

Cross Validation results saved to C:/temp/crossval.txt

Figure 5.7: This figure shows some options available in the 2CEE tool.

5.3 Industrial Evaluation of 2CEE Application

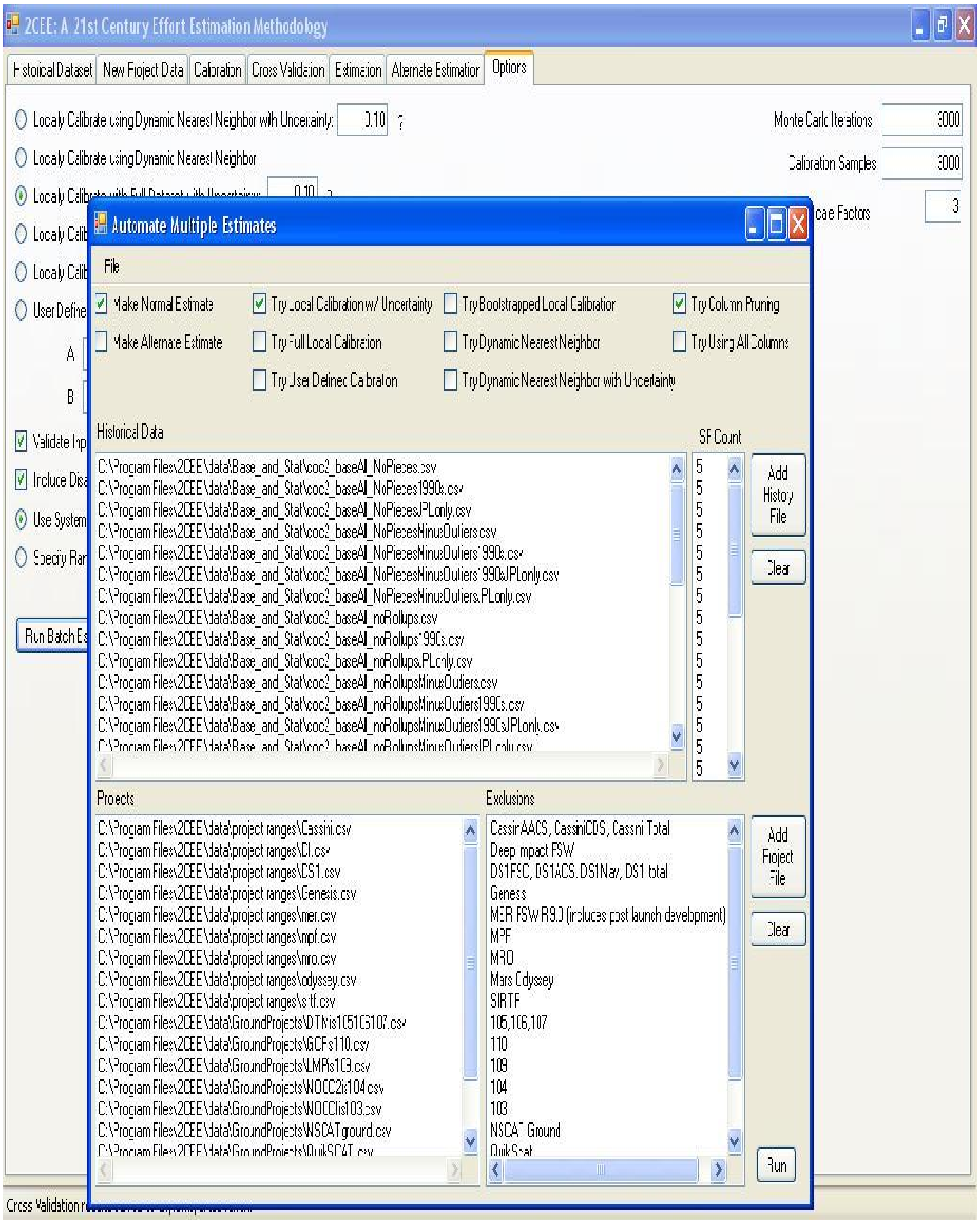
This section describes an experiment made to evaluate the performance of the 2CEE tool in the NASA JPL environment. The experiment was run using 2CEE's batch experiment mode. Scripts were then used to take the results from the log files created by 2CEE, and run them through the same evaluation scripts used in chapter 3. In particular, the nonparametric tests were done using Mann Whitney Wilcoxon rank-sum tests at a 95% confidence level. These evaluation scripts are available online at `http://unbox.org/wisp/var/dan/eval/`.

This experiment used a different dataset than the public *COC*81 or *NASA*93 datasets used earlier in this report. Instead, a proprietary dataset from JPL, *NASA*07, of 139 records was used for calibration and a specific set of 18 records were used for testing. Some of these test records were in *NASA*07 and the appropriate records were disabled from calibration when being estimated. The set of test records was carefully selected to represent the types of estimates JPL is commonly required to make. All records were in the COCOMO II format.

Several stratifications of the calibration data were used in this experiment. Some of them used only flight data or only ground data, and others both flight and ground. Some of the records in the dataset were smaller projects that were actually all part of one bigger project. This caused some of the stratifications to use the smaller pieces as records, and others instead used the rolled up projects. Some used only records from 1990 or sooner, some only JPL records instead of including records from other NASA centers, and some removed records considered to be outliers.

Many learners were included in this experiment, all of which were based on the COCOMO II model. These included the Boehm's default calibration, standard local calibration (section 2.3.2), JPL's SCAT model (section 2.3.2), and LOCOMO (section 2.7.2). It also included learners that applied some newer methods on top of either LC or LOCOMO. These additional methods included feature selection via COCOMIN (section 3.2.3), as well as the use of either bootstrapping or variance injection to represent the uncertainty in the historical data. These last two methods were not included with the expectation of improved accuracy. Rather, they were included to see if their use deteriorated accuracy or if it stayed the same.

Due to all of the stratification options tried with each learner, the total number of methods was quite large. Only selected results are shown here. The remaining results are freely available



2CEE: A 21st Century Effort Estimation Methodology

Historical Dataset | New Project Data | Calibration | Cross Validation | Estimation | Alternate Estimation | Options

( ) Locally Calibrate using Dynamic Nearest Neighbor with Uncertainty: 0.10 ?
( ) Locally Calibrate using Dynamic Nearest Neighbor
(●) Locally Calib
( ) Locally Calib
( ) Locally Calib
( ) User Define
A
B
[✓] Validate Inp
[✓] Include Dis
(●) Use System
( ) Specify Ra
Run Batch Es

Monte Carlo Iterations: 3000
Calibration Samples: 3000
cale Factors: 3

Cross Validation r

Automate Multiple Estimates

File

[✓] Make Normal Estimate
[ ] Make Alternate Estimate
[✓] Try Local Calibration w/ Uncertainty
[ ] Try Full Local Calibration
[ ] Try User Defined Calibration
[ ] Try Bootstrapped Local Calibration
[ ] Try Dynamic Nearest Neighbor
[ ] Try Dynamic Nearest Neighbor with Uncertainty
[✓] Try Column Pruning
[ ] Try Using All Columns

Historical Data	SF Count
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_NoPieces.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_NoPieces1990s.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_NoPiecesJPLonly.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_NoPiecesMinusOutliers.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_NoPiecesMinusOutliers1990s.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_NoPiecesMinusOutliers1990sJPLonly.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_NoPiecesMinusOutliersJPLonly.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_noRollups.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_noRollups1990s.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_noRollupsJPLonly.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_noRollupsMinusOutliers.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_noRollupsMinusOutliers1990s.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_noRollupsMinusOutliers1990sJPLonly.csv	5
C:\Program Files\2CEE\data\Base_and_Stat\coc2_baseAll_noRollupsMinusOutliersJPLonly.csv	5
	5

Add History File | Clear

Projects	Exclusions
C:\Program Files\2CEE\data\project ranges\Cassini.csv	CassiniAACS, CassiniCDS, Cassini Total
C:\Program Files\2CEE\data\project ranges\DI.csv	Deep Impact FSW
C:\Program Files\2CEE\data\project ranges\DS1.csv	DS1FSC, DS1ACS, DS1Nav, DS1 total
C:\Program Files\2CEE\data\project ranges\Genesis.csv	Genesis
C:\Program Files\2CEE\data\project ranges\mer.csv	MER FSW R9.0 (includes post launch development)
C:\Program Files\2CEE\data\project ranges\mpf.csv	MPF
C:\Program Files\2CEE\data\project ranges\mro.csv	MRO
C:\Program Files\2CEE\data\project ranges\odyssey.csv	Mars Odyssey
C:\Program Files\2CEE\data\project ranges\sirtf.csv	SIRTF
C:\Program Files\2CEE\data\GroundProjects\DTMis105106107.csv	105,106,107
C:\Program Files\2CEE\data\GroundProjects\GCFis110.csv	110
C:\Program Files\2CEE\data\GroundProjects\LMPis109.csv	109
C:\Program Files\2CEE\data\GroundProjects\NOCC2is104.csv	104
C:\Program Files\2CEE\data\GroundProjects\NOCCIis103.csv	103
C:\Program Files\2CEE\data\GroundProjects\NSCATground.csv	NSCAT Ground
C:\Program Files\2CEE\data\GroundProjects\QuikSCAT.csv	QuikScat

Add Project File | Clear | Run

Figure 5.8: This figure shows 2CEE's batch estimation mode. This can be used to create estimates from multiple methods and datasets in order to evaluate their performance.

at `http://unbox.org/wisp/var/dan/extras/appendix.pdf`. Note that in the results Fuzzy refers to the variance injection method for representing uncertainty described in section 4.4.1, and FSS refers to the COCOMIN feature selector described in section 3.2.3.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
NoPieces-Locomo-FSS	110	19	0	19	36.86	22.63	0.50	0.715
NoPieces-LC-FSS	121	8	0	8	53.26	33.43	0.39	0.603
NoPieces-Fuzzy	129	0	0	0	57.01	27.89	0.50	0.483
NoPieces-LC	129	0	0	0	56.93	28.78	0.50	0.489
NoPieces-Bootstrap	129	0	0	0	56.80	28.58	0.50	0.484
NoPieces-Locomo	129	0	0	0	56.24	28.86	0.50	0.494
SCAT	129	0	0	0	73.41	42.11	0.33	0.342
Boehm	128	0	1	-1	66.06	42.14	0.33	0.429

Table 5.1: Selected Results from JPL Experiment on All Data

Table 5.1 highlights some of the results. This table shows methods that calibrated with a stratification sample that used all of the data and chose rollups over pieces. The only method to have a nonparametric loss is Boehm's default calibration. However, the parametric measures tell a much different story. The use of LOCOMO and COCOMIN produced nearly half of the median MRE of SCAT and the default COCOMO. Although MMRE is not a good evaluation measure especially in this domain, it is worthy to note that the combination of LOCOMO and COCOMIN produced a much lower MMRE than the other methods. On this stratification sample, LOCOMO (without COCOMIN), COCOMIN (without LOCOMO), LC, and both uncertainty representation methods scored about the same.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
NoPieces-LC-FSS	176	7	0	7	31.82	14.19	0.44	0.709
NoPieces-Locomo-FSS	177	6	0	6	31.16	5.43	0.44	0.717
NoPieces-Bootstrap-FSS	177	6	0	6	31.95	15.63	0.44	0.733
NoPieces-Fuzzy-FSS	177	6	0	6	32.18	15.64	0.44	0.704
NoPieces-LC	183	0	0	0	35.74	21.97	0.56	0.736
NoPieces-Locomo	183	0	0	0	34.96	22.14	0.56	0.735
Boehm	183	0	0	0	40.25	29.10	0.44	0.717
SCAT	183	0	0	0	40.44	29.29	0.44	0.731
f_noRollups-LC-FSS	183	0	0	0	53.83	37.37	0.22	0.602

Table 5.2: Selected Results from JPL Experiment on Flight Data

Table 5.2 highlights the performance of some of the methods estimating for records of flight based projects. On the flight records, the best performing methods used COCOMIN. The use of COCOMIN and LOCOMO produced a very low median MRE of 5.4%! Once again, row and feature selection had superior parametric measures compared to default COCOMO and SCAT. One of the better performing combinations that used a flight stratification was included in this table showing that it had worse parametric scores of every kind.

Method	Ties	Wins	Losses	Win-Loss	MMRE	MedMRE	Pred30	R
g_NoPieces-Locomo	181	38	0	38	34.44	15.94	0.67	0.686
NoPiecesJPL-Locomo	192	27	0	27	31.05	20.08	0.67	0.813
g_NoPieces1990s-Locomo-FSS	199	20	0	20	33.45	26.81	0.67	0.729
NoPieces-Locomo-FSS	203	16	0	16	42.56	22.57	0.56	0.744
g_noRollups1990s-LC	216	3	0	3	44.35	28.87	0.44	0.738
SCAT	218	0	1	-1	106.38	42.11	0.22	0.645
Boehm	211	0	8	-8	91.87	42.14	0.22	0.653

Table 5.3: Selected Results from JPL Experiment on Ground Data

Table 5.3 highlights the performance of some of the methods estimating for records of ground based projects. On the ground records, the best performing methods used LOCOMO. The best scoring method used LOCOMO on ground records, and several other combinations using ground stratification also did well. The use of both COCOMIN and LOCOMO did well but not the best. Both default COCOMO and SCAT did poorly on the ground projects. They had losses in the nonparametric test, very high MMRE's, and significantly greater median MRE's.

Neither bootstrapping or ”fuzzy” variance injection greatly decreased performance. However, a keen observer will notice the very best performing methods did not use either technique. The hypothesis that these methods decrease performance is likely true, although the effect is slight.

Both LOCOMO and COCOMIN did well in this experiment. LOCOMO seemed better for ground estimates, and COCOMIN more suited for flight estimates. The use of both methods did well when estimating either flight or ground estimates.

In terms of stratification, using ground records only did have value when estimating ground projects, but the same was not true for flight data. The use of newer records (1990's and up) was sometimes useful for ground estimates. Overall, using all of the records and preferring rollups over pieces was a good choice.

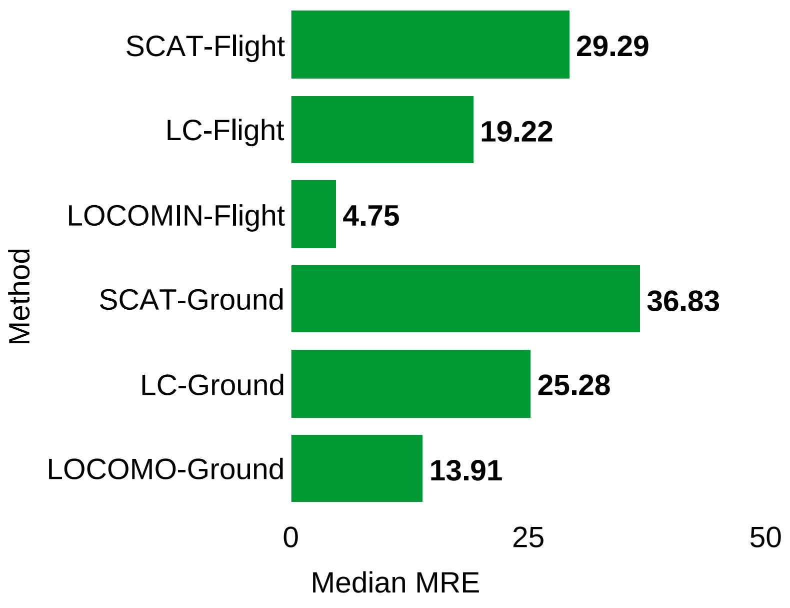


Figure 5.9: 2CEE vs Standard Methods

In the end, some significant improvements over current methods were found. As can be seen in figure 5.9, 2CEE can achieve improvements over an industrial state of the art, while simultaneously providing other improvements. For example, 2CEE provides automated local calibrations and uncertainty representation.

5.4 Unexpected Results

In the course of building and testing the 2CEE application, the use of its bootstrapping operation to measure uncertainty in the historical data was used. This operation results in a range of COCOMO a, b pairs which are the coefficients COCOMO uses to relate code size to effort. The literature [8, 10] reports a fairly confined variance seen in these variables. The results seen in 2CEE were much, much different from this expected norm.

These unexpected results showing very high COCOMO coefficient variation are shown in figure 5.10. In addition to the much higher range, the a coefficient was observed to be much higher than usual, and the b coefficient was much lower. This implies that the NASA domain has much better economies of scale than usual, but also much higher startup costs. This suggests they will be

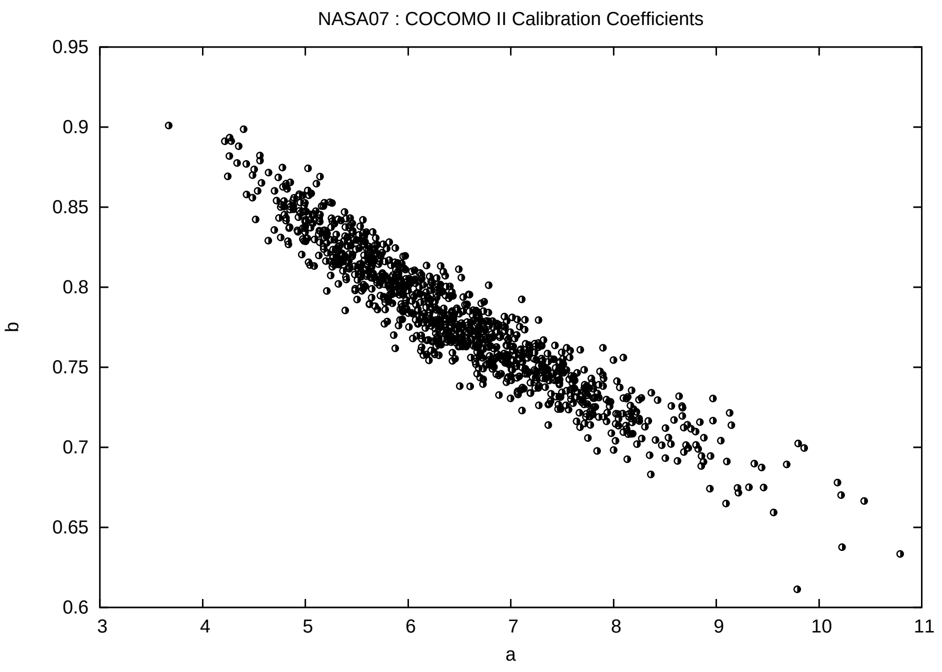


Figure 5.10: COCOMO II coefficients from local calibration of 1,000 bootstrap samples of the NASA07 dataset. Note that this is COCOMO II format so the *b* coefficient will be increased by the scale factors. A set of nominal scale factors will raise *b* by 0.1897.

more efficient with large projects and less efficient with small ones than many other organizations.

After observing this unexpected result on the proprietary NASA dataset, the same operation was tried on the public COCOMO datasets. Similar results were found: The range of coefficient values was much greater than expected, and the NASA data had a higher *a* and lower *b* coefficient.

5.5 Feedback from Industry

Beyond significantly improving estimates especially for ground software (see figure 5.9), 2CEE was found to be useful in an industrial setting for more practical reasons. For example, Karen Lum, a cost analyst at the Jet Propulsion Laboratory, comments:

> ”It used to take me a week to perform local calibration. I had to figure out which data points were outliers, which weren't. I had to try different stratification scenarios, etc. All this took a long time. Now, with 2CEE, I can perform local calibration for multiple

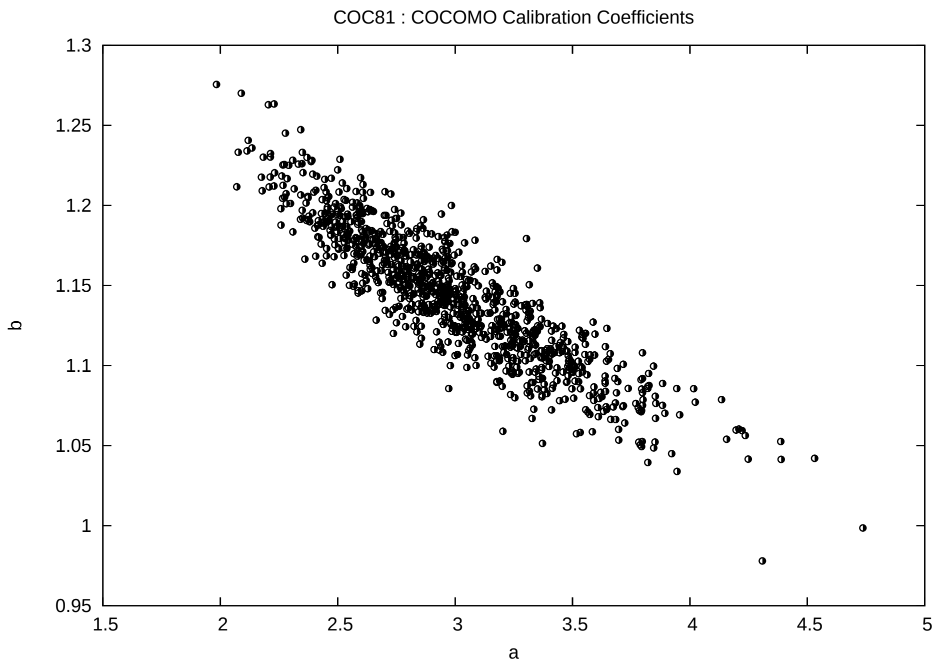


Figure 5.11: COCOMO coefficients from local calibration of 1,000 bootstrap samples of the NASA93 dataset. This dataset is in COCOMO 81 format so the *b* value will not be raised by any scale factors.

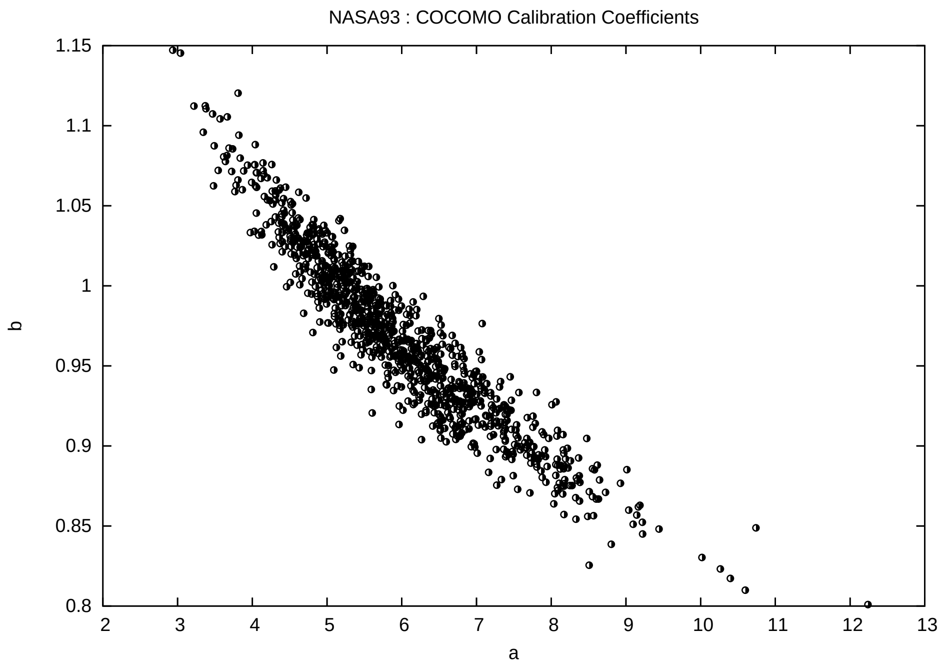


Figure 5.12: COCOMO coefficients from local calibration of 1,000 bootstrap samples of Boehm's COC81 dataset. This dataset is in COCOMO 81 format so the *b* value will not be raised by any scale factors.

> scenarios in minutes. I can slice and dice the data in a number of different ways and see my local calibration results right away. Using 2CEE saves me time so that I can focus on other aspects of estimation, such as analyzing the technical characteristics of the software project."

Some of this analysis of technical characteristics of the data, such as the calibration analysis shown previously in figure 5.10, is available directly in 2CEE. Unfortunately, this case study wasn't long enough to receive feedback regarding all of the tool's features such as the benefit of a greater representation of uncertainty.

Chapter 6

Conclusion

Recall the research goals for this thesis:

1. make progress towards the, ”conceptual framework of integration” of expert and model-based methods suggested by Meli [86]
2. implement the expert-judgment ”best practices” proposed by Jorgensen [49] in a model
3. provide methodology improvements that are both useful and used by industrial effort estimators, and to report feedback concerning the method’s real-life use
4. represent estimation uncertainty by accounting for Kitchenham’s sources of estimation error [68]
5. reduce the uncertainty of the estimate
6. provide estimation methods useful at different software life-cycle stages
7. improve estimation accuracy
8. explore the use of feature selection for software effort estimation
9. explore the use of meta-learners such as bagging and boosting for software effort estimation
10. evaluate methods using the nonparametric evaluation discussed in our previous work [95]

Each of these goals was accomplished, although some only partially. The following subsections summarize the accomplishments towards each goal.

Progressed Towards a Conceptual Framework of Integration of Expert and Model-Based Methods

Rather than starting from scratch with a expert and model hybrid system, this thesis added some characteristics of expert-judgment methods to a model to make progress towards integrating the two. Progress was made in two ways. First, a more open model-based framework that allows more expert involvement was created. Second, many of Jorgensen's expert judgment best practices were included in the model framework. This progress towards a hybrid approach to expert and model based effort estimation was implemented in a tool, 2CEE, and additional paths for integration are described.

Implemented Many of Jorgensen's Expert-Judgment Best Practices in a Model

The following seven expert estimation best practices from the original list of Jorgensen's 12 [49] were implemented in the 2CEE tool:

1. evaluate estimation accuracy, but avoid high evaluation pressure
2. ask the estimators to justify and criticize their estimates
3. avoid irrelevant and unreliable estimation information
4. use documented data from previous development tasks
5. assess the uncertainty of the estimate
6. provide feedback on estimation accuracy
7. find estimation experts with relevant domain background

The first of these seven practices was handled by using a cross validation experiment of the historical data with the model in question to generate a range of relative error values. This avoided high evaluation pressure by taking the human out of the loop. The second practice was handled both inherently by the use of a model calibrated on local data, the use of the well known and respected COCOMO model, as well as the cross validation previously mentioned. The third practice was accomplished by the use of feature selection algorithms, such as COCOMIN, and record selection algorithms, such as LOCOMO. These algorithms remove irrelevant and unreliable data from use. The fourth practice was accomplished by use of the historical dataset used to calibrate the model. The fifth practice was accomplished by representing the uncertainty in the project ranges with triangular distributions, the uncertainty in the historical data either through bootstrapping or variance injection, and the uncertainty from model error by stretching the estimate by the median MRE from the cross validation, and finally providing a range of estimates to the analyst. The sixth practice may be accomplished by making an estimate for a project in which you don't know the actual effort, and then having the actual effort revealed to you. This can be done much easier and faster with the 2CEE models than with expert-judgment. The seventh practice, "find estimation experts with relevant domain background", is accomplished by calibrating good models with relevant project data.

Provided Methodology Improvements to Industry and Reported Feedback

Recall that in a recent review of software effort estimation studies, Jorgensen and Shepperd discovered a, "lack of in-depth studies on the actual use of estimation methods and real-life evaluations" [52]. This thesis responds to Jorgensen and Shepperd's startling discovery by providing method improvements in a new tool called 2CEE and describing its reception at its client, the NASA Jet Propulsion Laboratory. Although this thesis is not a thorough review of the real-life use of software effort estimation, it is a step in the right direction.

The 2CEE tool is currently being phased into use at the Jet Propulsion Laboratory. During this process, it will be used alongside traditional methods until it has a proven track record. Afterwards, the tool will be used to make estimates. 2CEE is currently available for free from the Jet Propulsion Laboratory. To obtain a copy, send an email to `softwarerelease@jpl.nasa.gov` with your name, citizenship, affiliation, and the name of the software that you are requesting, 2CEE.

Represented Estimation Uncertainty

The problem of uncertainty in software effort estimates is explained in section 2.5.4. In 2CEE, the uncertainty of an estimate is represented by providing a sorted range of estimates. This range is plotted to produce an estimate curve such as is seen in Figure 5.6 on page 90.

To the best of our knowledge, this thesis describes the first implementation that can handle the model error, measurement error, and assumption error sources of uncertainty described by Kitchenham and Linkman [68]. Measurement error and assumption error are handled in 2CEE by the use of project ranges, bootstrapping, and variance injection on the calibration data. Model error is handled in 2CEE by performing leave-one-out cross validation, taking the median MRE score, and stretching the estimate curve accordingly.

Kitchenham and Linkman's last source of uncertainty, scope error, or the "error from estimating outside the model's domain", was not represented in 2CEE. This because of the inherent difficulty in quantifying this error, as well as the different practices involved. That is, a cost analyst won't calibrate to local data when a drastically domain difference occurs, but rather they will use a recommended industry-wide calibration such as Boehm's default COCOMO calibration [10]. On a related note, when using local data for calibration, automated nearest neighbor row selection algorithms such as LOCOMO [46, 95] should mitigate scope error by selecting the records closest to the novel domain.

Reduced Estimate Uncertainty

Properly using the methods suggested in this thesis will likely result in a greater degree of uncertainty shown in the estimate than a practitioner is used to seeing. This is because the uncertainty was being underrepresented because not all of its sources were considered. Despite this, 2CEE has the potential to reduce the true uncertainty of an estimate through use of feature and record selection techniques such as COCOMIN and LOCOMO.

Why do data pruning methods such as COCOMIN and LOCOMO reduce uncertainty? One reason lies in the potential uncertainty introduced by each additional feature and record that may have been collected incorrectly. By eliminating unnecessary information, the impact of human error is reduced. Another reason is that methods with improved accuracy will have less model error, and consequently less range in the estimate. Finally, automated nearest neighbor row selection algorithms such as LOCOMO may mitigate scope error, resulting in less uncertainty.

Provided Estimation Methods Useful Throughout the Software Development Life-cycle

Recall that Molokken-Ostvold et al. [101] found that the, ”estimate often changes over the course of a project, depending on the stage at which the estimate is made.” Thus, it is important to provide a tool useful for making estimates under different conditions and at different stages of development.

Because new projects are represented by ranges of values in 2CEE, estimates may be made very early in the software life-cycle when little is known about the new project. Estimates at this time will have a much greater uncertainty than normally, but they may be done. As time passes and more information is gained about the new project, the ranges of the values describing the project are restricted, creating a better estimate.

Improved Estimation Accuracy

The use of feature subset selection algorithms such as COCOMIN, and row selection techniques such as LOCOMO, were routinely found to be as good as or sometimes far better than traditional methods. For example, an experiment evaluating 2CEE in the JPL environment found that using LOCOMO and COCOMIN when used to estimate flight projects produced a 5.4% median MRE, whereas standard LC produced a 21.97% median MRE, and SCAT provided a 29.29% median MRE (section 5.2). Using LOCOMO to estimate ground-based projects resulted in a 15.9% median MRE, whereas standard LC and uncalibrated models such as SCAT produced 28.9% and 42.1% median MRE’s, respectively (section 5.3). These median MRE improvements are illustrated in figure 6.1.

As can be seen in figure 6.1, 2CEE can achieve improvements over an industrial state of the art such as SCAT, while simultaneously providing the other improvements mentioned in this chapter. For example, 2CEE provides automated local calibrations and more comprehensive uncertainty representation than SCAT.

Based on the large differences in gains from feature subset selection on COC81 data vs NASA93 data (section 3.2), it appears the potential accuracy gain is dependent on the organization involved. Validation of any novel method’s effectivity is suggested using within-company data.

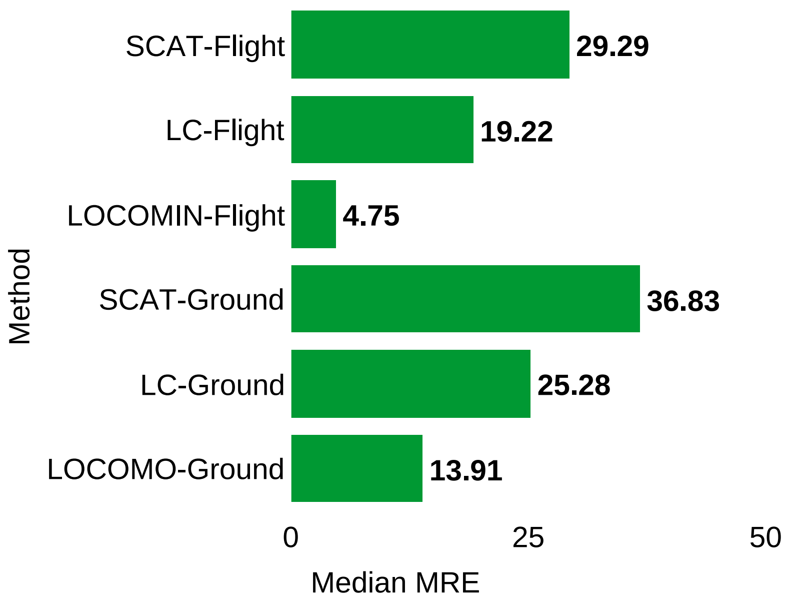


Figure 6.1: 2CEE vs Standard Methods

Explored Feature Subset Selection Methods for Software Effort Estimation

Research on feature subset selection, FSS, was a major portion of this thesis. Section 2.6 describes FSS in the literature. Section 3.2 describes experiments performed using FSS for the COCOMO model.

Feature selection was useful for the NASA93 dataset but not for the COC81 dataset. The FSS experiments found a near-linear time search of the feature space, COCOMIN, was just as effective as an exhaustive exploration of all feature combinations, COCOMOST. Many variations of the FSS learners were tried with different evaluation criteria, and for COCOMIN, with different search criteria. The most effective COCOMIN combination for the NASA93 dataset evaluated subsets using MMRE, pre-sorted the features by the ones with lowest associated MMRE when used to make estimates by themselves with KLOC, and tried both a forward and backward search, choosing the best evaluated subset at the end.

Explored Bagging and Boosting Techniques for Software Effort Estimation

The meta-learner techniques of bagging, described in section 2.8.2, and boosting, described in section 2.8.3, were studied with the use of the COCOMO model. Sometimes the use of the feature selection algorithm COCOMIN was included in these experiments. The details and results of these studies may be found in section 3.3 and section 3.4 respectively.

The majority of the use of bagging and boosting did not show significant improvement in model accuracy. In some of these cases, a very small improvement was observed. However, such a small gain is not statistically significant given the noise in the data. Furthermore, even if such a gain is a real effect, it comes at a much higher computational cost than basic COCOMO or COCOMIN. In contrast with most of these results, boosting by subsampling showed more significant improvement on the NASA93 data. A variant which used 10 boosting iterations of 75% subsamples of the COCOMIN learner reported a median MRE about 11% lower than standard LC (see 3.24). This result suggests subsampling may be useful in certain cases.

These results by no means rule out the potential benefit of bagging or more traditional boosting techniques such as Adaboost on effort estimation models. Rather they show that, except for boosting via subsamples, the configurations of boosting and bagging used did not show major improvement of the COCOMO model on the data available.

Evaluated Software Effort Estimation Methods with Nonparametric Techniques

Our previous work reports that the effort estimation errors do not fit a Gaussian distribution, but rather have a small number of very large outliers [95]. Having a Gaussian distribution is a critical assumption of most parametric measures. We suggested using nonparametric evaluation methods, particularly the Mann Whitney Wilcoxon rank-sum test [83]. More details on nonparametric evaluation is mentioned in section 2.5.3.

The experiments detailed in this thesis were evaluated using the Mann Whitney Wilcoxon rank-sum test [83]. We suggested that, ”since we seek methods that can be rejected, the value of interest to us is how often methods *lose* [46, 95].” The evaluations made frequently found several methods without any losses. Thus, parametric measures such as median MRE were used as a secondary criteria evaluation measure.

Unexpected Results

The research into uncertainty in the input data led to some interesting results. The literature [8, 10] reports a fairly confined variance seen in these variables of a linear coefficient between 2.5 and

2.94, and a scalar coefficient between 0.91 and 1.01. However, results from the industry (see figure 5.10), find a linear coefficient between 4 and 11, and a scalar coefficient between 0.65 and 0.9 This implies that the NASA domain has much better economies of scale than usual, but also much higher startup costs. This suggests they will be more efficient with large projects and less efficient with small ones than many other organizations.

After observing this unexpected result on the proprietary NASA dataset, the same operation was tried on the public COCOMO datasets. Similar results were found: The range of coefficient values was much greater than expected, and the NASA data had a higher linear and lower scalar coefficient.

These results are a warning. The models aren't as stable as commonly believed. Small changes in the calibration data can create large changes in the calibration coefficients. This is probably due to the scarce data problem described in section 2.5.2. Assuming this to be the cause, three ways to achieve more stable calibrations include:

1. gather more beneficial calibration records
2. reduce the number of features used in the model
3. improve the precision of metric collection

Thus, the COCOMIN feature selector described in section 3.2.3 may be used to increase the stability of the calibration coefficients.

6.1 Future Work

Further work should be done to bridge the gap between expert and model based software effort estimation techniques. In addition, the human side of using estimation models needs to be researched.

6.1.1 Stacking Estimation Methods

Future work should be done in both researching combining different estimation methods, as well as adding this ability to the 2CEE tool. Ideally each component method should be as different as possible but also accurate. One way to combine methods, such as by the COSEEKMO tool by Menzies et al. [46, 93], is to internally evaluate each method on the available data and use the best performing one to make the estimate. Alternately, instead of combination schemes like COSEEKMO, the estimates of each method can be combined.

Although average votes and weighted averages work to combine point estimates, combining a range is a little more complicated. The simplest way to combine a range of estimates such as those made by 2CEE is to first have each model produce their own ranges of estimates. Then make one big list of all of the estimates from each model, sort it, and plot it.

Future work could also be done to combine direct expert judgment based estimates with model-based estimates. This is achievable in much the same way as combining the estimates of separate models. However, in a framework like 2CEE, the expert estimates would simply need to be converted into a list of 1,000 or more estimates before being combined with the model estimates. If the expert estimate is a point estimate, the expert must assess the uncertainty and then this list of estimates can be made.

6.1.2 Implement More Best Practices in 2CEE

Although the following best practices were not implemented, it would be straightforward to implement them in a tool like 2CEE.

- combine estimates from different experts and estimation strategies
- estimate top-down and bottom-up, independently of each other
- use estimation checklists
- provide estimation training opportunities

The first of these could be done by using a stacking approach of estimation models. The second practice would require a lot of development to create the bottom-up assessment tools, but it could be done. The third of these practices could be implemented easily by requiring steps be taken in the application. This constraint was relaxed instead in favor of more flexibility. This leaves the responsibility of following a checklist to the cost analyst. Finally, providing estimation training opportunities could be built into a model as an interactive training guide.

The only expert estimation best practice which cannot translate into into direct inclusion in a model, but must instead be handled by the experts using the model, is to, ”avoid conflicting estimation goals.” The expert is ultimately responsible for maintaining their integrity when using a model, and must avoid tweaking the model to get an estimate more fitting to any inappropriate motives.

6.1.3 Include More Models in 2CEE

Future work should be done to include additional models in the 2CEE tool. Currently, the tool only has COCOMO based models. It would be useful to include some non COCOMO models in the tool to get another point of view. In addition, there are other COCOMO based algorithms that might be added. For example, the COSEEKMO tool by Menzies et al. [46,93] is an ideal candidate to add to 2CEE.

6.1.4 Interactive Feature Exploration

While implementing the feature selection methods in 2CEE, an idea for an interactive feature selection and exploration tool was conceived but there was not enough time to create it. Such a tool should offer data visualization and analysis of the features. For example, histograms of the feature’s distribution, as well as statistics on entropy and deviation could be included.

The interactive feature selection portion of this tool would list the current feature set with a bunch of related metrics from a cross validation experiment. It would also include the related metrics from each feature subset that is one step away from the current set, i.e. either one feature removed or one feature added from the current set. The metrics available could include standard parametric measures such as median MRE, MMRE, Pred, correlation, etc. It could also include a nonparametric test such as the Mann Whitney Wilcoxon rank-sum test: [83], comparing the current set of features against the same model using each of the other sets of features that are one step away from the current set.

To use the interactive feature selection, the analyst would consider all of the analysis measures, and would guide a search through the feature space. This would be done by either choosing to keep the current set of features, or selecting one of the subsets which is one feature different from the current set to become the new current set. Then the entire list of metrics is repopulated to show the new current set and each of the subsets which is one step away.

The steps of this tool are very similar to the operation of a feature selection algorithm. The benefit of including a human in the loop instead of automating the entire process is that a human mind can consider all of the related metrics when making a decision. Conversely, in an algorithm must choose an evaluation method and stick to it. This is most commonly done with a single parametric measure. Although an automated algorithm could be written to simulate the decision making of a human analyzing all of the related metrics, this would be a very complex task.

In addition, the list of subsets could be sortable by each metric allowing the analyst to observe which set is more likely to minimize a given measure.

6.1.5 Tradeoff Analysis

Features for tradeoff analysis should be included in the 2CEE methodology and implemented in the tool. An intuitive way this could be done is to allow multiple labeled graphs to be plotted on the same chart.

Bibliography

[1] Certified parametric practitioner tutorial. In *Proceedings of the 2006 International Conference of the International Society of Parametric Analysts, Seattle, WA*, 2006.

[2] A. Aamodt and E. Plaza. Case-based reasoning; foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, March 1994. Available from `http://www.iiia.csic.es/People/enric/AICom_ToC.html`.

[3] L. Angelis, I. Stamelos, and M. Morisio. Building a software cost estimation model based on categorical data. In *METRICS '01: Proceedings of the 7th International Symposium on Software Metrics*, page 4, Washington, DC, USA, 2001. IEEE Computer Society.

[4] S. Asch. Studies of independence and conformity: A minority of one against a unanimous majority. *Psychological Monographs: General and Applied*, 70(9):1970, 1956.

[5] John Bailey. Using monte carlo and cocomo-2 to model a large it system development, 2002.

[6] David A. Bell and Hui Wang. A formalism for relevance and its application in feature subset selection. *Mach. Learn.*, 41(2):175–195, 2000.

[7] Alberto Bertoni, Paola Campadelli, and M. Parodi. A boosting algorithm for regression. In *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks*, pages 343–348, London, UK, 1997. Springer-Verlag.

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

[9] B. Boehm. Safe and simple software cost analysis. *IEEE Software*, pages 14–17, September/October 2000. Available from `http://www.computer.org/certification/beta/Boehm_Safe.pdf`.

[10] 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.

[11] Petronio L. Braga, Adriano L. I. Oliveira, and Silvio R. L. Meira. Software effort estimation using machine learning techniques with robust confidence intervals. In *HIS 2007: 7th International Conference on Hybrid Intelligent Systems 2007*, pages 352–357, 2007.

[12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Technical report, Wadsworth International, Monterey, CA, 1984.

[13] L. Briand, K. Eman, and F. Bomarius. Cobra: A hybrid method for software cost estimation, benchmarking, and risk assessment, 1998.

[14] Lionel C. Briand, Victor R. Basili, and Christopher J. Hetmanski. Developing interpretable models with optimized set reduction for identifying high-risk software components. *IEEE Trans. Softw. Eng.*, 19(11):1028–1044, 1993.

[15] Lionel C. Briand, Victor R. Basili, and William M. Thomas. A pattern recognition approach for software engineering data analysis. *IEEE Trans. Softw. Eng.*, 18(11):931–942, 1992.

[16] Lionel C. Briand, Khaled El Emam, Dagmar Surmann, Isabella Wieczorek, and Katrina D. Maxwell. An assessment and comparison of common software cost estimation modeling techniques. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 313–322, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[17] Lionel C. Briand, Khaled El Emam, and Isabella Wieczorek. A case study in productivity benchmarking: Methods and lessons learned.

[18] Rich Caruana and Dayne Freitag. Greedy attribute selection. In *International Conference on Machine Learning*, pages 28–36, 1994.

[19] Zhihao Chen, Tim Menzies, Dan Port, and Barry Boehm. Finding the right data for software cost modeling. *IEEE Software*, Nov 2005.

[20] Zhihoa Chen, Tim Menzies, and Dan Port. Feature subset selection can improve software cost estimation. In *Proceedings, PROMISE workshop, ICSE 2005*, 2005. Available from `http://menzies.us/pdf/05/fsscocomo.pdf`.

[21] 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.

[22] Alexander Yun chung Liu. The effect of oversampling and undersampling on classifying imbalanced text datasets. Master's thesis, 2004. Available from `http://www.lans.ece.utexas.edu/~aliu/papers/aliu_masters_thesis.pdf`.

[23] Iris Fabiana de Barcelos Tronto, Jose Demisio Simoes da Silva, and Sant'Anna Nilson. Comparison of artificial neural network and regression models in software effort estimation. In *IJCNN 2007: International Joint Conference on Neural Networks 2007*, pages 771–776, 2007.

[24] Sarah Jane Delany, Pdraig Cunningham, and Wolfgang Wilke. The limits of cbr in software project estimation.

[25] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006. Availiable from `http://jmlr.csail.mit.edu/papers/v7/demsar06a.html`.

[26] Orsolya Doban and Andras Pataricza. Cost estimation driven software development process.

[27] Harris Drucker. Improving regressors using boosting techniques. In *Proc. 14th International Conference on Machine Learning*, pages 107–115. Morgan Kaufmann, 1997.

[28] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

[29] D. Ferens and D. Christensen. Calibrating software cost models to Department of Defense Database: A review of ten studies. *Journal of Parametrics*, 18(1):55–74, November 1998.

[30] Gavin R. Finnie and Gerhard E. Wittig. Ai tools for software development effort estimation. In *SEEP '96: Proceedings of the 1996 International Conference on Software Engineering: Education and Practice (SE:EP '96)*, page 346, Washington, DC, USA, 1996. IEEE Computer Society.

[31] T. Fos, E. Stensrud, B. Kitchenham, and I. Myrtveit. A simulation study of the model evaluation criterion mmre. *IEEE Transactions on Software Engineering*, 29(11):985 – 995, November 2003.

[32] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS: Journal of Computer and System Sciences*, 55, 1997.

[33] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[34] J. Friedman. Greedy function approximation: a gradient boosting machine, 1999.

[35] Liu Fuyan. An attribute selection approach and its application. In *ICNNB 2005 International Conference on Neural Networks and Brain*, pages 636–640, 2005.

[36] Stein Grimstad, Magne Jorgensen, and Kjetil Molokken-Ostvold. Software effort estimation terminology: The tower of babel. *Information and Software Technology*, 48(4):302–310, 2006.

[37] Tanja Gruschke and Magne Jorgensen. Assessing uncertainty of software development effort estimates: The learning from outcome feedback. In *METRICS '05: Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS'05)*, page 4, Washington, DC, USA, 2005. IEEE Computer Society.

[38] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1998.

[39] 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. Available from `http://www.cs.waikato.ac.nz/~mhall/HallHolmesTKDE.pdf`.

[40] Mark A. Hall. Selection of attributes for modeling bach chorales by a genetic algorithm.

[41] Mark A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proc. 17th International Conf. on Machine Learning*, pages 359–366. Morgan Kaufmann, San Francisco, CA, 2000.

[42] F.J. Heemstra. Software cost estimation models. In *Information Technology, 1990. 'Next Decade in Information Technology', Proceedings of the 5th Jerusalem Conference on (Cat. No.90TH0326-9)*, pages 286–297, 1990.

[43] Jairus Hihn and Karen T. Lum. Improving software size estimates by using probabilistic pairwise comparison matrices. In *METRICS '04: Proceedings of the Software Metrics, 10th International Symposium on (METRICS'04)*, pages 140–150, Washington, DC, USA, 2004. IEEE Computer Society.

[44] J.M. Hihn and H. Habib-agahi. Cost estimation of software intensive projects: A survey of current practices. In *Proceedings of the Thirteenth IEEE International Conference of Software Engineering*, May 1991.

[45] O. Jalali, T. Menzies, D. Baker, and J. Hihn. Column pruning beats stratification in effort estimation. In *Proceedings, PROMISE workshop, Workshop on Predictor Models in Software Engineering*, 2007.

[46] Omid Jalali. Evaluation bias in effort estimation. Master's thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2007.

[47] R. Jeffery, M. Ruhe, and I. Wieczorek. Using public domain metrics to estimate software development effort. In *Proceedings of the 7th International Software Metrics Symposium*, pages 16–27, 2001. Available from `http://www.iese.fhg.de/pdf_files/iese-058_00.pdf`.

[48] T.C. Jones. *Estimating Software Costs*. McGraw-Hill, 1998.

[49] M. Jorgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1-2):37–60, 2004.

[50] M. Jorgensen. Practical guidelines for expert judgment based-software-effort estimation. *IEEE Software*, 22(3), May/June 2005.

[51] M. Jorgensen and S. Grimstad. Over-optimism in software development projects: ?the winner?s curse? In *Proceedings of IEEE CONIELECOMP, Puebla, Mexico, February 28-March 2*, pages 280–285. IEEE Computer Society, 2005.

[52] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies, January 2007. Available from `http://www.simula.no/departments/engineering/publications/Jorgensen.2005.12`.

[53] Magne Jorgensen. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Trans. Softw. Eng.*, 21(8):674–681, 1995.

[54] Magne Jorgensen. Realism in assessment of effort estimation uncertainty: It matters how you ask. *IEEE Trans. Softw. Eng.*, 30(4):209–217, 2004.

[55] Magne Jorgensen and K. Molokken-Ostvold. Reasons for software effort estimation error: Impact of respondent role, information collection approach, and data analysis method. *IEEE Transactions on Software Engineering*, 30(12):993–1007, 2004.

[56] Magne Jorgensen and Dag I. K. Sjoberg. Impact of experience on maintenance skills. *Journal of Software Maintenance*, 14(2):123–146, 2002.

[57] G. Kadoda, M. Cartwright, L. Chen, and M. Shepperd. Experiences using case based reasoning to predict software project effort, 2000.

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

[59] Chris F. Kemerer. Reliability of function points measurement: a field experiment. *Commun. ACM*, 36(2):85–97, 1993.

[60] J. Kernaghan and R. Cooke. The contribution of the group process to successful project planning in r&d settings. *IEEE Transactions on Engineering Management*, 33(3):134–140, 1986.

[61] Jacky Keung, Ross Jeffery, and Barbara Kitchenham. The challenge of introducing a new software cost estimation technology into a small software organization. In *ASWEC '04: Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04)*, page 52, Washington, DC, USA, 2004. IEEE Computer Society.

[62] Taghi M. Khoshgoftaar, Laurent Nguyen, Kehan Gao, and Jayanth Rajeevalochanam. Application of an attribute selection method to cbr-based software quality classification. In *ICTAI '03: Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, page 47, Washington, DC, USA, 2003. IEEE Computer Society.

[63] 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.

[64] C. Kirsopp, M. Shepperd, and J. Hart. Search heuristics, case-based reasoning and software project effort prediction, 2002.

[65] B. A. Kitchenham, E. Mendes, and G. H. Travassos. Cross- vs. within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering*, pages 316–329, May 2007.

[66] B.A. Kitchenham, L.M. Pickard, S.G. MacDonell, and M.J. Shepperd. What accuracy statistics really measure. *Software, IEE Proceedings*, 148(3):81–85, 2001.

[67] Barbara Kitchenham. A procedure for analyzing unbalanced datasets. *IEEE Trans. Softw. Eng.*, 24(4):278–301, 1998.

[68] Barbara Kitchenham and Stephen Linkman. Estimates, uncertainty, and risk. *IEEE Softw.*, 14(3):69–74, 1997.

[69] J.P.C. Kliijnen. Sensitivity analysis and related analyses: a survey of statistical techniques. *Journal Statistical Computation and Simulation*, 57(1–4):111–142, 19987.

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

[71] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[72] Linda M. Laird. The limitations of estimation. *IT Professional*, 8(6):40–45, 2006.

[73] G.V. Lashkia and L. Anthony. Relevant, irredundant feature selection and noisy example elimination. In *2004 IEEE International Conference on Man and Cybernetics Systems*, volume 34, pages 888–897, 2004.

[74] Aleksandar Lazarevic and Vipin Kumar. Feature bagging for outlier detection. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 157–166, New York, NY, USA, 2005. ACM Press.

[75] Roger J. Lewis. An introduction to classification and regression tree cart analysis, 2004. Available from `http://www.saem.org/download/lewis1.pdf`.

[76] Jingzhou Li and Guenther Ruhe. A comparative study of attribute weighting heuristics for effort estimation by analogy. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering*, pages 66–74, New York, NY, USA, 2006. ACM Press.

[77] Harold A. Linstone and Murray Turoff (eds). *The Delphi Method: Techniques and Applications*. Addison-Wesley, 1975.

[78] K. Lum, J. Powell, and J. Hihn. Validation of spacecraft cost estimation models for flight and ground systems. In *ISPA Conference Proceedings, Software Modeling Track*, May 2002.

[79] Karen Lum. Software cost analysis tool user document, 2005.

[80] Karen Lum, Michael Bramble, Jairus Hihn, John Hackney, Mori Khorrami, and Erik Monson. Handbook for software cost estimation, 2003.

[81] Stephen G. MacDonell and Martin J. Shepperd. Comparing local and global software effort estimation models – reflections on a systematic review. In *ESEM 2007: First International Symposium on Empirical Software Engineering and Measurement, 2007.*, pages 401–409. IEEE Computer Society, 2007.

[82] Carolyn Mair and Martin J. Shepperd. The consistency of empirical comparisons of regression and analogy-based software project cost prediction. In *ISESE*, pages 509–518, 2005.

[83] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18(1):50–60, 1947. Available on-line at http://projecteuclid.org/DPubS?service=UI&version=1.0&verb=Display&handle=euclid.aoms/1177730491.

[84] Cuauhtemoc Lopez Martin, Jerome Leboeuf Pasquier, Cornelio M. Yanez, and Agustin T. Gutierrez. Software development effort estimation using fuzzy logic: A case study. In *ENC '05: Proceedings of the Sixth Mexican International Conference on Computer Science*, pages 113–120, Washington, DC, USA, 2005. IEEE Computer Society.

[85] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent in function space, 1999.

[86] Roberto Meli. Human factors and analytical models in software estimation: An integration perspective. In *Proceedings of the ESCOM-SCOPE 2000*, pages 33–40, Munich, Germany, 2000. Shaker Publishing.

[87] T. Menzies, O. Elrawas, D. Baker, J. Hihn, and K. Lum. On the value of stochastic abduction (if you fix everything, you lose fixes for everything else). In *International Workshop on Living with Uncertainty (an ASE'07 co-located event)*, 2007. Available from http://menzies.us/pdf/07fix.pdf.

[88] T. Menzies, O. Elwaras, J. Hihn, Feather M, B. Boehm, and R. Madachy. The business case for automated software engineering. In *IEEE ASE*, 2007. Available from http://menzies.us/pdf/07casease-v0.pdf.

[89] T. Menzies and J. Hihn. Evidence-based cost estimation for better quality software. *IEEE Software*, July/August 2006. Available on-line at http://menzies.us/pdf/06costs.pdf.

[90] T. Menzies, K. Lum, and J. Hihn. The deviance problem in effort estimation. 2006. Available from http://menzies.us/06deviations.pdf.

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

[92] 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.

[93] Tim Menzies, Zhihao Chen, Jairus Hihn, and Karen Lum. Selecting best practices for effort estimation. *IEEE Transactions on Software Engineering*, November 2006. Available from http://menzies.us/pdf/06coseekmo.pdf.

[94] 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.

[95] Tim Menzies, Omid Jalali, Jairus Hihn, Dan Baker, and Karen Lum. Software effort estimation and conclusion stability, 2007.

[96] A. Miller. *Subset Selection in Regression (second edition)*. Chapman & Hall, 2002.

[97] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[98] K. Molokken and M. Jorgensen. A review of surveys on software effort estimation. In *ISESE'03*, 2003.

[99] Kjetil Molokken and Magne Jorgensen. Expert estimation of web-development projects: Are software professionals in technical roles more optimistic than those in non-technical roles? *Empirical Softw. Eng.*, 10(1):7–30, 2005.

[100] Kjetil Molokken-Ostvold and Magne Jorgensen. Group processes in software effort estimation. *Empirical Softw. Eng.*, 9(4):315–334, 2004.

[101] Kjetil Molokken-Ostvold, Magne Jorgensen, Sinan S. Tanilkan, Hans Gallis, Anette C. Lien, and Siw E. Hove. A survey on software estimation in the norwegian industry. In *METRICS '04: Proceedings of the Software Metrics, 10th International Symposium on (METRICS'04)*, pages 208–219, Washington, DC, USA, 2004. IEEE Computer Society.

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

[103] Petr Musílek, Witold Pedrycz, Giancarlo Succi, and Marek Reformat. Software cost estimation with fuzzy models. *SIGAPP Appl. Comput. Rev.*, 8(2):24–29, 2000.

[104] Petr Musilek, Witold Pedrycz, Nan Sun, and Giancarlo Succi. On the sensitivity of cocomo ii software cost estimation model. In *METRICS '02: Proceedings of the 8th International Symposium on Software Metrics*, page 13, Washington, DC, USA, 2002. IEEE Computer Society.

[105] Ingunn Myrtveit and Erik Stensrud. A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Trans. Softw. Eng.*, 25(4):510–525, 1999.

[106] Ingunn Myrtveit, Erik Stensrud, and Martin Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Transactions on Software Engineering*, 31(5):380–391, May 2005.

[107] Chee Un Ng and Graham R. Martin. Automatic selection of attributes by importance in relevance feedback visualization. In *IV '04: Proceedings of the Information Visualization, Eighth International Conference on (IV'04)*, pages 588–595, Washington, DC, USA, 2004. IEEE Computer Society.

[108] Ursula Passing and Martin Shepperd. An experiment on software project size and effort estimation. In *ISESE '03: Proceedings of the 2003 International Symposium on Empirical Software Engineering*, page 120, Washington, DC, USA, 2003. IEEE Computer Society.

[109] Parag C. Pendharkar, Girish H. Subramanian, and James A. Rodger. A probabilistic model for predicting software development effort. *IEEE Trans. Softw. Eng.*, 31(7):615–624, 2005.

[110] Alfred M. Pietrasanta. Current methodological research. In *Proceedings of the 1968 23rd ACM national conference*, pages 341–346, New York, NY, USA, 1968. ACM.

[111] G. Ridgeway, D. Madigan, and T. Richardson. Boosting methodology for regression problems, 1999.

[112] G. C. Roper-Lowe and J. A. Sharp. The analytic hierarchy process and its application to an information technology decision. *The Journal of the Operational Research Society*, 41(1):49–59, 1990.

[113] Saharon Rosset. Robust boosting and its relation to bagging. In *KDD '05: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 249–255, New York, NY, USA, 2005. ACM Press.

[114] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[115] B. Boehm S. Chulani, B. Clark and B. Steece. Calibration approach and results of the cocomo ii post-architecture model. In *Proceedings ISPA,98*, 1998.

[116] T.L. Saaty. *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill, New york, 1980.

[117] R. Schapire. The boosting approach to machine learning: An overview, 2001.

[118] M. Shepperd. Software project economics: A roadmap. In *International Conference on Software Engineering 2007: Future of Software Engineering*, 2007.

[119] M. Shepperd and Gada F. Kadoda. Comparing software prediction techniques using simulation. *IEEE Trans. Software Eng*, 27(11):1014–1022, 2001.

[120] M. Shepperd and C. Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12), November 1997. Available from `http://www.utdallas.edu/~rbanker/SE_XII.pdf`.

[121] Martin Shepperd and Michelle Cartwright. Predicting with sparse data. *IEEE Trans. Softw. Eng.*, 27(11):987–998, 2001.

[122] Martin Shepperd, Chris Schofield, and Barbara Kitchenham. Effort estimation using analogy. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, pages 170–178, Washington, DC, USA, 1996. IEEE Computer Society.

[123] Surendra K. Singhi and Huan Liu. Feature subset selection bias for classification learning. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 849–856, New York, NY, USA, 2006. ACM Press.

[124] D.P. Solomatine and D.L. Shrestha. Adaboost.rt: A boosting algorithm for regression problems. In *2004 IEEE International Joint Conference on Neural Networks*, volume 2, pages 1163–1168. IEEE, 2004.

[125] K. Srinivasan and D. Fisher. Machine learning approaches to estimating software development effort. *IEEE Trans. Soft. Eng.*, pages 126–137, February 1995.

[126] Ioannis Stamelos, Lefteris Angelis, Maurizio Morisio, Evaggelos Sakellaris, and George L. Bleris. Estimating the development cost of custom software. *Inf. Manage.*, 40(8):729–741, 2003.

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

[128] E. Stensrud and I. Myrtveit. Human performance estimating with analogy and regression models: An empirical validation. In *METRICS '98: Proceedings of the 5th International Symposium on Software Metrics*, page 205, Washington, DC, USA, 1998. IEEE Computer Society.

[129] Kevin Strike, Khaled El Emam, and Nazim H. Madhavji. Software cost estimation with incomplete data. *Software Engineering*, 27(10):890–908, 2001.

[130] R. Strutzke. *Estimating Software-Intensive Systems: Products, Projects and Processes*. Addison Wesley, 2005.

[131] Liang Tian and Afzel Noore. Multistage software estimation. *Proceedings of the 35th Southeastern Symposium on System Theory*, pages 232–236, 2003.

[132] Adam Trendowicz, Bernhard Graser, and Ernst Haunschmid. Optimal project feature weights in analogy-based cost estimation: Improvement and limitations. *IEEE Trans. Softw. Eng.*, 32(2):83–92, 2006. Member-Martin Auer and Member-Stefan Biffl.

[133] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features, 2001.

[134] Isabella Wieczorek and Melanie Ruhe. How valuable is company-specific data compared to multi-company data for software cost estimation? In *METRICS '02: Proceedings of the 8th International Symposium on Software Metrics*, page 237, Washington, DC, USA, 2002. IEEE Computer Society.

[135] Ian H. Witten and Eibe Frank. *Data mining. 2nd edition*. Morgan Kaufmann, Los Altos, US, 2005.

[136] Lei Yu and Huan Liu. Efficient feature selection via analysis of relevance and redundancy. *J. Mach. Learn. Res.*, 5:1205–1224, 2004.