# Empirically-Guided Software Effort Guesstimation

**Philip M. Johnson** **Carleton A. Moore**
**Joseph A. Dane** **Robert S. Brewer**
Collaborative Software Development Laboratory
Information & Computer Sciences Department
University of Hawaii
Honolulu, Hawaii 96822 USA
johnson@hawaii.edu

**Abstract**
Software project effort estimation is frequently seen as complex and expensive for individual software engineers. In Project LEAP, we are investigating tools and methods to support low-cost, empirically-based software developer improvement. In a recent case study, we investigated effort estimation and the relative accuracy of a dozen different analytic estimation procedures. Student programmers could estimate the effort required using any of the analytic methods, or else provide their own "guesstimate". Our study provides evidence that "guesstimates", when informed by low-cost analytical methods, may be the most accurate of all.

**Keywords:** software estimation, software quality, software planning

## 1 Introduction

Most software engineers (and their managers) fervently wish they could get home from the office earlier at night and come in less on weekends. Typically, escaping the tyranny of crunch mode requires greater control over the development process, and reasonably accurate project estimates provide essential support for acquiring this control.

Given the undisputable lifestyle benefits that can result, it is ironic that many developers continue to view project estimation as a kind of software engineering version of the "np-complete problem". In other words, some fear that creating an accurate project estimate might be, in the worst case, as costly as simply building the system itself.

*Appearing in IEEE Software, November 2000*

The state of the art in software project estimation does not deserve this reputation, but the fact remains that modern methods are time-consuming and complex. Estimation methods such as COCOMO II [3] are oriented toward the needs of large and very large software engineering projects for creating estimates of cost, effort, and schedule. In COCOMO II, an organization provides values characterizing organizational factors such as "precedentedness", "development flexibility", "team cohesion", and "process maturity" as part of the estimation process. On the other hand, methods such as the Personal Software Process (PSP) [8] support small/individual project estimation accuracy. In the PSP, individuals collect personal data concerning software size, effort, and defects for their own code-level work products, then use a regression-based analytic technique called PROBE to generate effort predictions for future projects based upon historical data. Although both of these approaches have demonstrated success in organizations with the resources necessary to adopt them, the process overhead involved in their implementation can often be inconsistent with the resource-constrained nature of smaller or even startup level development organizations.

For several years, we have pursued an initiative called Project LEAP, whose goal is the improvement of individual developers though lightweight, empirical, anti-measurement dysfunction, and portable software engineering tools and methods [10]. One result of this project is the LEAP toolkit, a publically available, Java-based suite of applications for collection and analysis of an individual's software engineering data. Among other things, LEAP contains tools to simplify the collection of size data (at the level of lines of code, methods, and classes) and effort data (in developer minutes). The collected size and effort data serves as input to a set of estimation tools that can produce over a dozen different analytical estimates of the effort required for a new project given an estimate of its size, using various estimation methods such as linear, logarithmic, or exponential regressions. During project planning, the developer can review and select one of the estimates produced by the analytical tools, or else substitute their own "guesstimate" based upon their own experience and review of the analytical estimates.

During the Fall of 1999, we performed a case study using the LEAP toolkit in a graduate software engineering class. One of the goals of the study was to evaluate the various analytical estimation methods made available by the toolkit. We were curious as to whether a single method would prove most accurate, or whether the most accurate method would depend upon the type of project or the specific developer. To our surprise, we found that, in most cases, the developer-generated "guesstimates" were more accurate than the analytical estimates. We also found that the PROBE method of the Personal Software Process [8], perhaps the most widely publicized analytical approach to personal effort estimation, was the sixth most accurate method. Finally, we found that access to a range of analytical estimation methods appeared to be useful to developers in generating their guesstimates and improving them over time.

Due to the small number of participants in the case study, not all of our results are statistically significant. Replication is necessary to better understand its generality and applicability. However, our initial findings do support some provocative conjectures concerning the research and practice of project estimation. First, much of the research on project estimation focuses on the evaluation of a single estimation method. Success is often demonstrated by increased estimation accuracy over time. Our study suggests that more research should be done in which a variety of different estimation methods are investigated simultaneously, since even suboptimal estimation methods may exhibit improvement over time. Second, our research suggests that a practical approach to small-project software estimation might be "empirically guided guesstimation", whereby a variety of simple analytical methods inform developer intuition to create a low-cost yet useful software project estimates.

The remainder of this article elaborates on these ideas. The next section briefly describes how estimation is performed in the LEAP toolkit. The following section presents selected results from the case study. The final section provides some recommendations for future research and practice.

## 2 Estimation using the LEAP toolkit

The LEAP toolkit provides a suite of tools for collecting and analyzing personal software engineering data (see Sidebar). For the purposes of this article, only the tools relating to size collection, effort collection, and project estimation are relevant.

Effort collection in LEAP is straightforward: tools allow developers to enter effort data either after the fact or interactively while they work. Size collection uses a tool called LOCC [4] that counts non-comment source lines, methods, and classes for any language for which a JavaCC-compliant grammar is available. Unlike most other size counting tools, LOCC also includes a "diff" facility which allows the developer to count the number of source lines, methods, and classes that *changed* between two versions of the system, which is critical for useful estimation in an evolutionary development setting. Together, these tools provide cross-platform, cross-language support for low-cost size and effort collection.

Producing an estimate of the effort required for a new software project using LEAP involves the following basic steps:

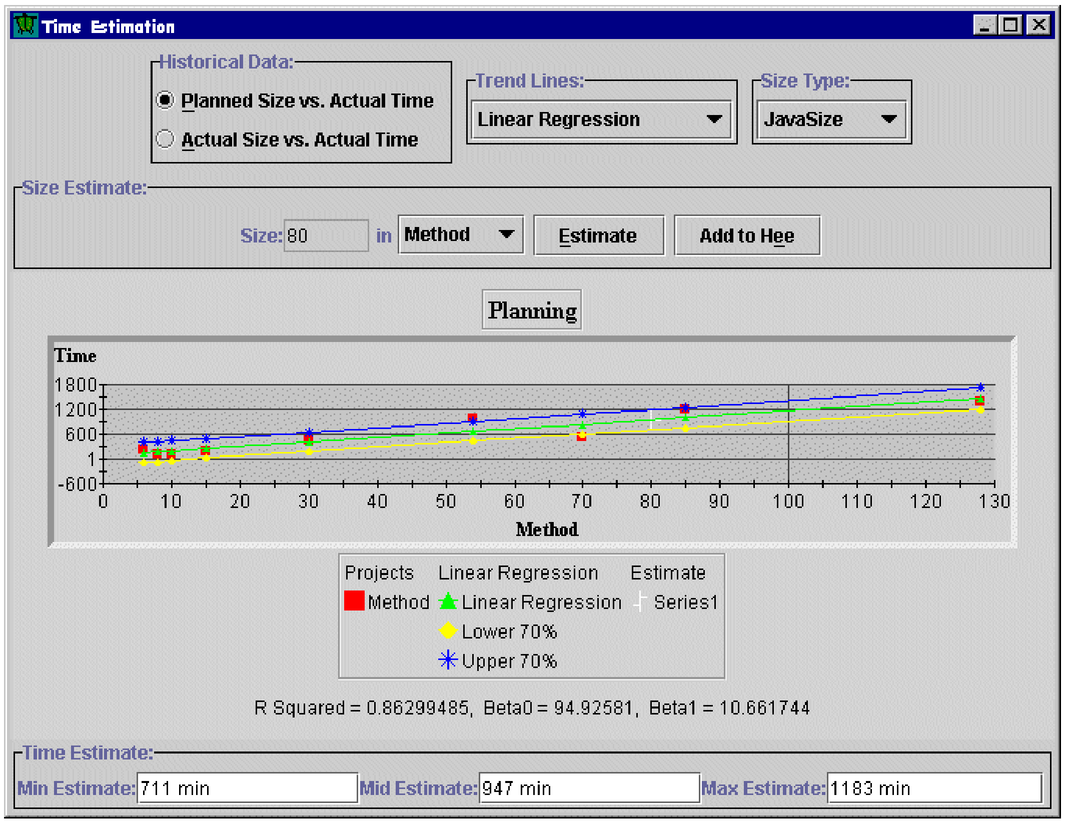
1. Select the completed projects whose size and effort data you wish to use as input to the analytical estimation methods.

2. Generate an estimate of the expected size (in lines, methods, or classes).

3. Browse the effort estimates produced by the various analytical methods.

4. Record an effort estimate to use, either from those proposed by the analytical methods or by generating your own "guesstimate".

Steps three and four are illustrated in the accompanying figures. Figure 1 shows the LEAP interface to the analytical effort estimation method browser. In the top panel, the user can indicate the type of historical data to be used (planned or actual), the trend line (i.e. the type of analytical model to be applied to the data), and the size metric (in this case, Java-based lines, methods, and classes). The displayed graph and the three estimates



Historical Data: [✓ Planned Size vs. Actual Time] Actual Size vs. Actual Time

Trend Lines: Linear Regression

Size Type: JavaSize

Size Estimate: Size: 80 in Method

| Projects | Linear Regression | Estimate |
|---|---|---|
| 🟥 Method | ▲ Linear Regression | ┼ Series1 |
| | ◆ Lower 70% | |
| | ✳ Upper 70% | |

R Squared = 0.86299485, Beta0 = 94.92581, Beta1 = 10.661744

Time Estimate:
Min Estimate: 711 min
Mid Estimate: 947 min
Max Estimate: 1183 min

Figure 1: The LEAP tool interface for effort estimation

in the bottom panel indicate the results of estimating the effort required to produce an 80 method software system, based upon eight prior software projects, using linear regression.

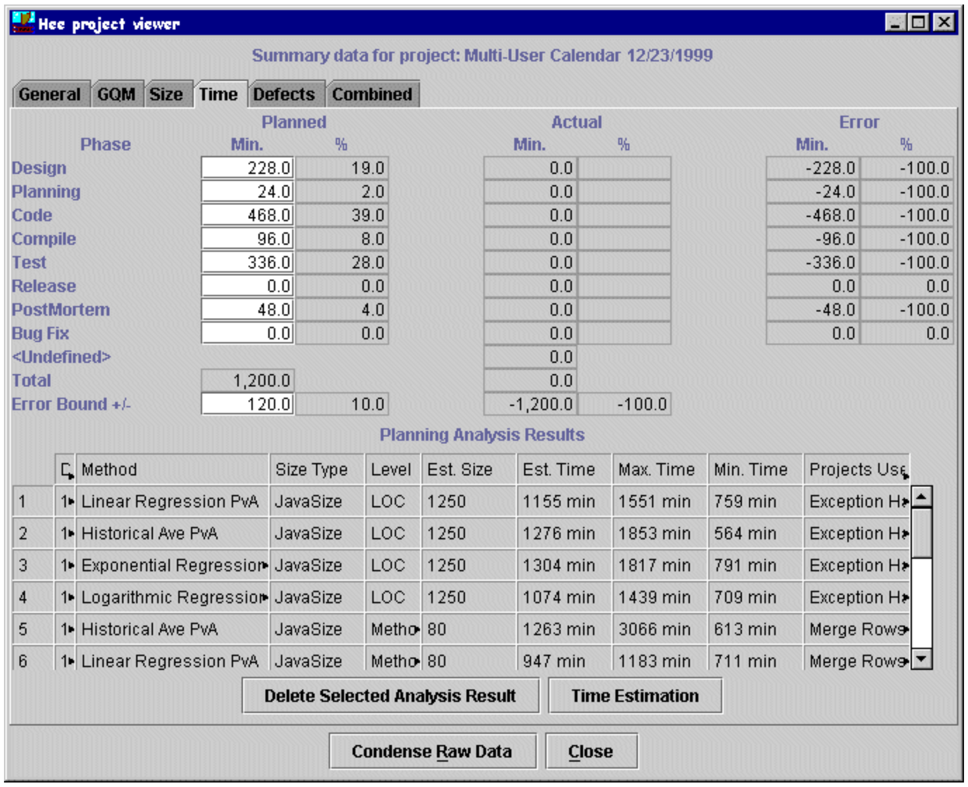
Choosing different trend lines and size metrics enables the developer to generate and then save a variety of estimates. Figure 2 shows one pane in the LEAP project planning tool. The upper half of the window provides fields in which the developer enters their planned effort. The lower half of the window shows a table in which the developer can collect together for review a set of estimates generated using the effort estimation tool. (The “Add to Hee” button in Figure 1 adds the data currently displayed to the table in Figure 2.) As you can see in Figure 2, the analytical effort estimates range in value from 947 minutes to 1304 minutes, yet the developer has entered a planned effort of 1200 minutes, which differs from all of the analytically-derived estimates.

The LEAP toolkit allows substantial flexibility in the kinds of work products and size metrics involved in estimation. While LOCC provides “shrink-wrapped” support for counting object-oriented programming languages like Java and C++, LEAP provides a size metric definition mechanism that allows users to integrate new measures for new document types. For example, the size of a high level requirements document might be counted in pages, while a design specification might be counted in function points.

The ability of LEAP to support multiple estimation approaches and represent planned and actual efforts for projects enabled us to explore an interesting research question: would users pick the most accurate estimate, and if so, which estimation technique would it be? The following case study provides some initial insight into this question.

## 3 Results from a case study

To gain some initial data concerning estimation using the LEAP toolkit, we conducted a case study in a graduate software engineering class of 16 students at the University of Hawaii. Seven of the 16 students were “experienced” software developers, with five or more years of prior programming experience. The 16 students developed 8 software projects each for a total of



Summary data for project: Multi-User Calendar 12/23/1999

| | Planned | | Actual | | Error | |
|---|---|---|---|---|---|---|
| Phase | Min. | % | Min. | % | Min. | % |
| Design | 228.0 | 19.0 | 0.0 | | -228.0 | -100.0 |
| Planning | 24.0 | 2.0 | 0.0 | | -24.0 | -100.0 |
| Code | 468.0 | 39.0 | 0.0 | | -468.0 | -100.0 |
| Compile | 96.0 | 8.0 | 0.0 | | -96.0 | -100.0 |
| Test | 336.0 | 28.0 | 0.0 | | -336.0 | -100.0 |
| Release | 0.0 | 0.0 | 0.0 | | 0.0 | 0.0 |
| PostMortem | 48.0 | 4.0 | 0.0 | | -48.0 | -100.0 |
| Bug Fix | 0.0 | 0.0 | 0.0 | | 0.0 | 0.0 |
| <Undefined> | | | 0.0 | | | |
| Total | 1,200.0 | | 0.0 | | | |
| Error Bound +/- | 120.0 | 10.0 | -1,200.0 | -100.0 | | |

Planning Analysis Results

| | C | Method | Size Type | Level | Est. Size | Est. Time | Max. Time | Min. Time | Projects Use |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Linear Regression PvA | JavaSize | LOC | 1250 | 1155 min | 1551 min | 759 min | Exception H |
| 2 | 1 | Historical Ave PvA | JavaSize | LOC | 1250 | 1276 min | 1853 min | 564 min | Exception H |
| 3 | 1 | Exponential Regression | JavaSize | LOC | 1250 | 1304 min | 1817 min | 791 min | Exception H |
| 4 | 1 | Logarithmic Regression | JavaSize | LOC | 1250 | 1074 min | 1439 min | 709 min | Exception H |
| 5 | 1 | Historical Ave PvA | JavaSize | Metho | 80 | 1263 min | 3066 min | 613 min | Merge Rows |
| 6 | 1 | Linear Regression PvA | JavaSize | Metho | 80 | 947 min | 1183 min | 711 min | Merge Rows |

Figure 2: One pane in the LEAP project plan tool. The table in the lower half of the screen lists the estimates used in coming up with the plan.

128 projects. However, the students used the first three projects to generate an initial set of historical data, and then practiced the estimation technique described above on the remaining five projects. Thus, only the last five projects by the 16 students, or 80 projects total were used to generate the following results concerning estimation.

Over the course of this semester, the students improved in their estimation capabilities in a manner similar to the results obtained from other small-project empirical methods such as the Personal Software Process [7, 5, 6, 9]. For example, the average student size estimate was off by approximately 50% on the third project but decreased to less than 15% by the eighth project. Similarly, the average effort estimate was off by approximately 25% on the third project but decreased to less than 10% on average on the eighth project.

What is novel about this case study is not that the students became better at estimation, which is by now a well-established property of curricula of this type. What is novel is the ability to investigate the relative accuracy of the estimates chosen relative to the other estimates that were available. In other words, did the students pick the right estimate to use amongst those that were available? To assess this, we calculated the error in the estimates by subtracting the planned value from the actual value, and performed an analysis of variance (ANOVA) test to see if the average error for one approach was statistically different from another approach. We interpret the results as statistically significant if the results could be due to chance less than 2% of the time ($p < .02$). More details are available in [10].

The results from the case study are provocative. First, over the course of the 80 projects in which estimation was performed, an analytical estimate was used as the student's planned effort estimate less than 10% of the time. Students vastly preferred to enter their own "guesstimate" rather than use one of the analytical values, although they typically recorded around three analytical estimates before entering their guesstimate. For 8 out of 16 students, these "guesstimates" were on average more accurate (i.e. had the smallest average error) than any of the analytical estimates. For two of the eight students, the guesstimates were significantly more accurate than any of the analytical methods ($p < .02$). For the other six students, their guesstimates were more accurate on average but not significantly more accurate than the next most accurate method from a statistical point of view. Interestingly, there was only one student for which an analytical technique (exponential regression using actual LOC) was the best estimate overall and significantly better than any other estimation technique ($p < .02$).

When the average relative error is computed for each estimation method over the class as a whole, it reveals that the student guesstimates were significantly more accurate than any of the analytical methods ($p < 0.001$). The next most accurate estimation technique (which was also, incidentally, significantly more accurate than any of the other analytical estimation techniques) was exponential regression using actual methods.

The PROBE analytical method from the Personal Software Process is perhaps the most widely researched current method for individual effort estimation, and so we were particularly interested to see how it fared against the other methods. For one student, the PROBE method was the most accurate estimation method on average (although not significantly more accurate than the next most accurate method from a statistical point of view). However, for the other 15 students, their own guesstimates were more accurate on average than the PROBE estimate. When the class data is viewed as a whole, the PROBE method was the sixth most accurate method.

## 4 Lessons Learned

One must be careful when interpreting these results. Even though some of our results are statistically significant, the teaching method, development environment, choice of programming projects, and other factors greatly influence the data and outcomes. There are, of course, a myriad of differences between an academic and professional development environment that might influence such data. That said, we present the following as the major lessons we believe can be learned from our experiences with estimation in the LEAP toolkit:

1. **Software process data collection is costly. Even more automated support than that currently provided by the LEAP toolkit will be useful.** Collecting effort and size data, even with the advanced tool support provided by LEAP, is still viewed by developers as a distraction from the task at hand, even when the future benefits to them are clear. As a result, we are currently investigating "ultra-lightweight" project estimation support, in which deep integration with a specific development environment could provide almost total automation of effort and size data collection for personal small-project estimation.

2. **Providing multiple estimation techniques and size measures adds value.** Our users found it interesting and useful to "browse" the various estimation methods to look for similarities and differences. In some cases, users would "triangulate" their estimate by choosing a value midway between several of the analytical estimates. In addition, while estimation based upon lines of code may be best for small projects, estimation based upon the number of methods may be preferable as the size of the system increases.

3. **Analytical estimation methods need not be complex.** The estimation methods included in LEAP range from the very simple (such as a method based upon average, minimum, and maximum productivity) to the relatively complicated (the PROBE method, in which the developer must choose between three analytical methods based upon the strength of correlation between planned and actual data). Our preliminary results suggest that complicated methods may not necessarily yield a more accurate estimate, particularly when developers can incorporate their own intuition into the estimate. During the postmortem interview on each project, students would often explain that they decided to deviate from the analytical estimates to account for various idiosyncracies in the project, their background, or other factors. Automating the breadth of knowledge brought to bear on the estimation problem by these users within an analytical estimation technique will be problematic at best.

4. **Empirical data is useful.** Although our users rarely adopted an analytical estimate without change, it was also clear that they found the data of tremendous utility as a way of "getting in the right ballpark." When a user's estimate departed radically from the analytical range of values, they

invariably had an excellent rationale for their decision.

The importance of empirical data is highlighted by an analysis of estimation accuracy between two groups in the case study: the "experienced" developers (those with five or more years of experience) and the "inexperienced" developers (those with less than five years of experience). To our surprise, we discovered that experienced developers substantially underestimated the effort required for their projects at the beginning of the study. Their guesstimates were so far off that they were not only worse than the analytic estimates, they were even less accurate than those of the inexperienced developers! It seems as though inexperienced developers appear to "trust" the empirical data and let it guide their guesstimates from the beginning, while experienced developers initially ignored the empirical data. However, this effect was temporary. By the end of the study, experienced developers seemed to rely more on the empirical data in forming their guesstimates, and their estimation accuracy improved to a level similar to that of the inexperienced developers.

## 5 Acknowledgements

This research was supported in part by a grant from the National Science Foundation (CCR-9804010).

## 6 Sidebar: The LEAP toolkit

"LEAP" is an acronym for four of the fundamental design principles underlying the toolkit. Lightweight, Empirical, Anti-measurement dysfunction, and Portable. *Lightweight* means that the tool does not force you (or force the colleagues that work with you) to conform to its own highly structured and/or constrained development process. *Empirical* means that the toolkit helps you to collect and analyze a variety of numerical measures which can help give you insight into your strengths and weaknesses. (Non-numerical, qualitative insight is also supported in the toolkit.) *Anti-measurement dysfunction* [1] means that the toolkit is designed with a recognition that attaching numbers to people can be a social or professional liability in certain organizations, and so the tool is designed to support privacy concerns. Finally, *portable* acknowledges the highly mobile nature of current software professionals both within and across company boundaries, and thus the need for personal data collection and analysis software that can move with the professional into different platforms, organizations, and application development domains.

LEAP consists of over a dozen integrated tools, supporting collection of defect data, effort data, size data, checklists, and patterns. LEAP also allows definition of work product types, defect types, severity levels, development phases, and size measures. The project planning tool supports size and effort estimation, GQM (goal-question-metric) [2] documentation about the nature and use of the data collected, and various analyses concerning productivity, defect rates, and so forth.

The Leap toolkit is written in Java. Since 1997, we have made over 30 public releases of the Leap toolkit. As of January, 2000, the Leap toolkit consisted of 44,000 lines of code, 2,209 methods, 287 classes, and 14 packages. You can download the latest version of the Leap toolkit from <http://csdl.ics.hawaii.edu/Tools/LEAP/LEAP.html>. The LOCC toolkit is also publically available, and can be downloaded from <http://csdl.ics.hawaii.edu/Tools/LOCC/LOCC.html>.

## REFERENCES

[1] R. D. Austin. *Measuring and Managing Performance in Organizations*. Dorset House Publishing, 1996.

[2] V. Basili and D. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, SE-10(6), November 1984.

[3] B. Boehm, B. Clark, E. Horowitz, R. Madachy, R. Selby, and C. Westland. Cost models for future software lifecycles: COCOMO 2.0. In *Annals of Software Engineering*. 1995.

[4] J. A. Dane. Modular program size counting. M.S. thesis, University of Hawaii, December 1999.

[5] K. E. Emam, B. Shostak, and N. Madhavji. Implementing concepts from the Personal Software Process in an industrial setting. In *Proceedings of the Fourth International Conference on the Software Process*, Brighton, England, December 1996.

[6] P. Ferguson, W. S. Humphrey, S. Khajenoori, S. Macke, and A. Matvya. Introducing the Per-

sonal Software Process: Three industry cases. *IEEE Computer*, 30(5):24–31, May 1997.

[7] W. Hayes and J. W. Over. The Personal Software Process (PSP): An empirical study of the impact of PSP on individual engineers. Technical Report CMU/SEI-97-TR-001, Software Engineering Institute, Pittsburgh, PA., 1997.

[8] W. S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley, New York, 1995.

[9] P. M. Johnson and A. M. Disney. A critical analysis of PSP data quality: Results from a case study. *Journal of Empirical Software Engineering*, December 1999.

[10] C. A. Moore. *Investigating Individual Software Development: An Evaluation of the Leap Toolkit*. PhD thesis, University of Hawaii, Department of Information and Computer Sciences, August 2000.