Lessons Learned from Teaching Reflective Software Engineering using the Leap Toolkit

Carleton A. Moore
Collaborative Software Development Laboratory
Information & Computer Sciences Department
University of Hawaii, Manoa
Honolulu, Hawaii 96822 USA
(808) 956-6920
cmoore@hawaii.edu

ABSTRACT

Keywords
Software Developer Education, Process Improvement, Measurement, Personal Software Process

1 Introduction

After using and teaching the Personal Software Process (PSP)[6] for over four years, we came to appreciate the insights and benefits that it gave us. However, we noticed some general problems with the PSP.

First, we started to question the quality of the data we recorded. In 1998, Anne Disney and Dr. Philip Johnson conducted a study to look at the data quality of the PSP data. They found that there are significant data quality issues with the manual PSP[2, 3].

Second, our experiences with industrial partners, management practices, and Robert Austin's book "Measuring and Managing Performance in Organizations"[1] brought to our attention the issue of measurement dysfunction in empirical software process improvement and review data. Essentially, measurement dysfunction means that an organization's goals to improve software development and formal technical review may inadvertently pressure their members to produce "good" metrics while actually reducing their performance.

These problems led us to begin work on an alternative software process improvement method called Reflective Software Engineering. Like the PSP, Reflective Software Engineering is based upon a simple idea: people learn best from their own experience.

Reflective Software Engineering supports experience-based improvement in developers' professional activities by helping the developer structure their experience, record it, and analyze it. Unlike the PSP, Reflective Software Engineering is designed around the presence of extensive automated support. The support is provided by a Java-based toolkit called "Leap" <http://csdl.ics.hawaii.edu/Research/LEAP/LEAP.html>. The kinds of structured insights and experiences users can record with Leap include: the size of the work product; the time it takes to develop it; the defects that the user or others find in it; the patterns that they discover during the development of it; checklists that they use during or design as a result of the project; estimates for time or size that they generate during the project; and the goals, questions, and measures that the user uses to motivate the data recording.

2 Overview of the Leap toolkit

The Leap toolkit implements the following four design principles.

- **Light-Weight** The first principle is that any tool or process used in software developer improvement should be light-weight. This means that neither tools nor processes should create substantial new work for the developer.
- **Empirical** The second principle is that the methods for software developer improvement should be empirical in nature. The improvements in the development process should be based upon the developer's experiences. By looking at their development empirically the developer can judge for themselves what is best.
- **Anti-measurement dysfunction** The third principle is that methods for developer improvement should address measurement dysfunction. If there is measurement dysfunction, then the data and analyses will not reflect reality. Any insights gained from the data and analyses will be faulty and may cause the developer to change their development practices in detrimental ways.
- **Portable** The fourth principle is that any tool or process used in software developer improvement should be portable. A tool that supports developer improvement but cannot follow the developer as they move to different organizations is not going to help that developer for very long.

The Leap toolkit is written in Java. Since 1997, we have made over 30 public releases of the Leap toolkit. As of

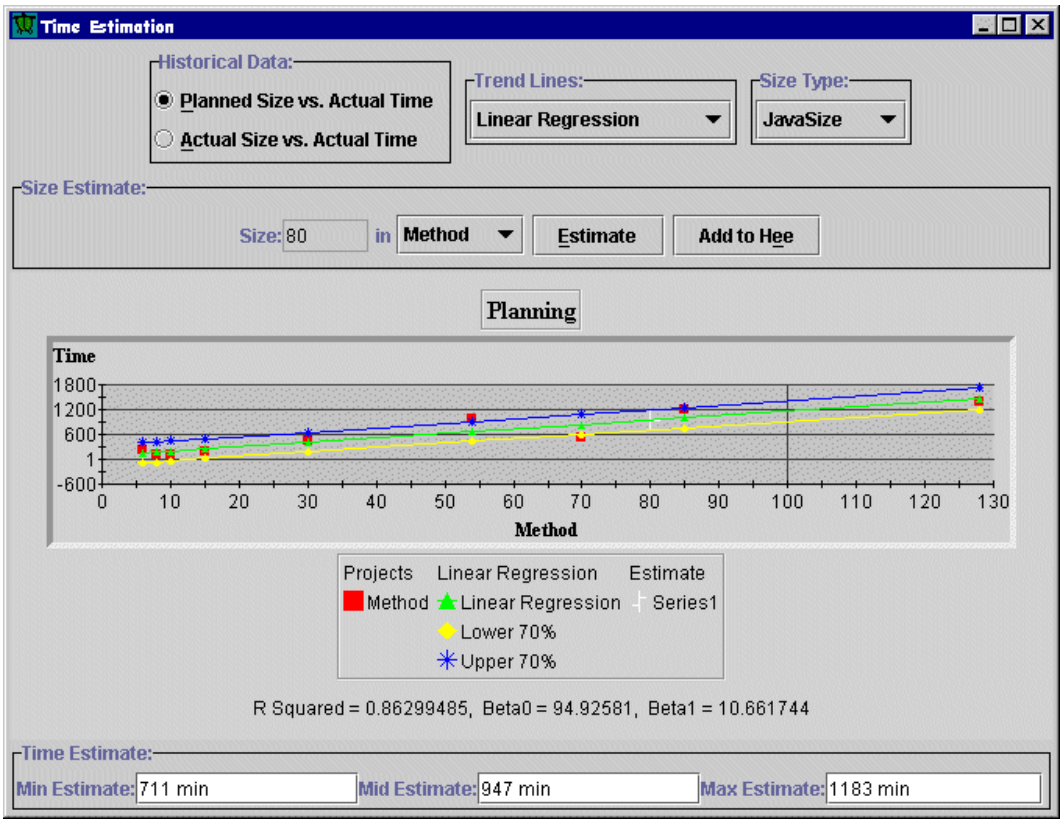

Figure 1: The Leap Time Estimation Tool.

January, 2000, the Leap toolkit consisted of 44,000 lines of code, 2,209 methods, 287 classes, and 14 packages. You can download the Leap toolkit from <http://csdl.ics.hawaii.edu/Tools/LEAP/LEAP.html>.

Figures 1 and 2 give a sense of the look and feel of the Leap toolkit we present . These figures show two of the dozen or so tools present in the toolkit. Figure 1 shows a screen-shot of the Leap Time Estimation Tool. The developer can choose the type of historical data, the trend lines, the size type and the estimated size to base their time estimate on. Once these items are selected, the Leap Time Estimation Tool calculates for the developer an estimated time range for the project.

Figure 2 shows a screen-shot of He'e, the Leap project summary tool. He'e allows the developer to quickly see an overview of a project. They can view the size, time and defect data and other summaries of their project. The developer can see how accurate their predictions were, their distribution of time between different development phases, when they injected and removed defects, and what their productivity was.

3 Lessons Learned

We have used the Leap toolkit for over two years in our research group, CSDL <http//csdl.ics.hawaii.edu>. In addition, we have used it in three software engineering courses to help teach software engineering principles. Although we have not actively solicited external usage, several industrial users have discovered, downloaded, and used it. They have generally reported favorable reactions to the Leap toolkit.

The following lessons arise from all of these experiences but, especially, from survey and interview data we collected from our most recent classroom usage. This class involved 16 students who used the Leap toolkit for four months in a graduate software engineering class.

Lesson 1: Collecting data about software development is useful

Many of the students felt that just collecting the data about their software development practice focussed their attention on the process. One student said "I'm focusing my attention more on the project. ... [It] makes me think I'm managing myself." Another student said "You pay attention to the project. So, you do [it] quickly."

By collecting the data and being aware of what phase they are in, the students started to change their software development practices. One student noted "I actually predict my phases, and do it [programming] in different phases. Before, I just do it together."

Lesson 2: Leap enables users to accurately estimate size and time in a known domain

The 16 students were able to accurately predict the size and

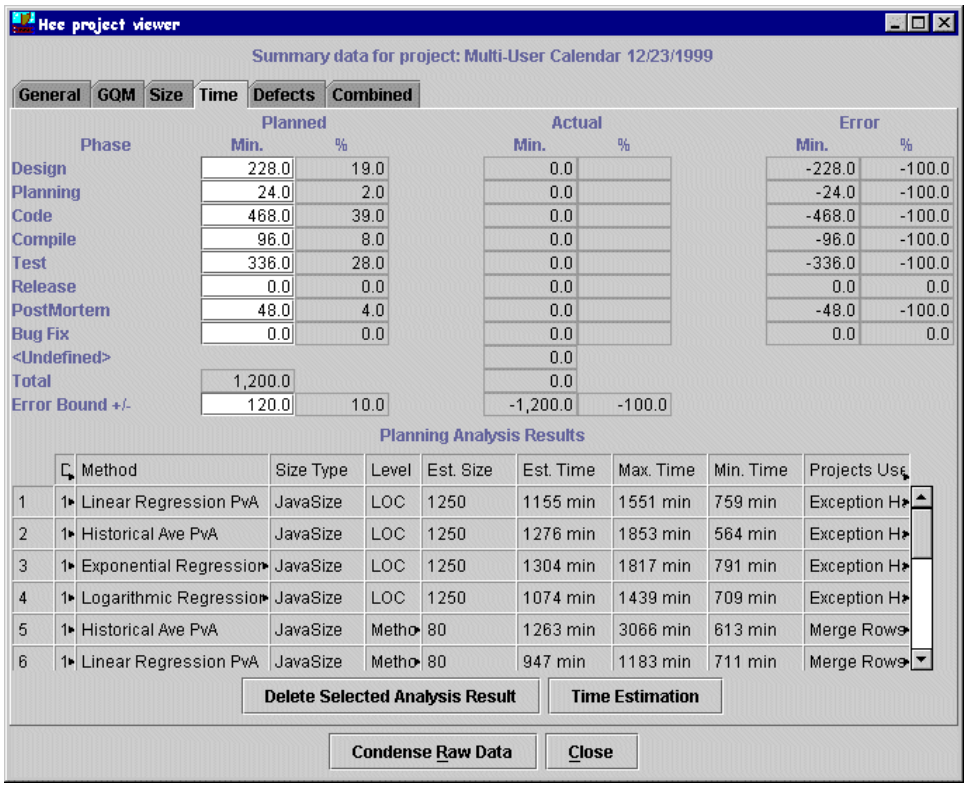

Figure 2: He'e, the Leap project summary tool showing the time summary.

effort required for their programming assignments. For program #3, only four students' time estimates were within 20% of their actual times. By program # 8, 12 students' time estimates were within 20% of their actual times. For their size estimates, only two students were within 20% accuracy for their fourth program. By the last program, seven students were within 20% for their size estimates. Tables 1 and 2 show the class' average estimation error for time and size. The size estimate is in methods. The last four projects were very similar, building and extending a GUI interface. The class' ability to estimate both size and time dramatically improved over the last four projects.

Table 1: Class' average time estimation error

Project					
3	4	5	6	7	8
-14.83	-17.45	-15.24	-19.40	-7.80	0.43

Table 2: Class' average size estimation error

Project					
3	4	5	6	7	8
-38.44	-24.58	-31.47	-34.570	-25.57	7.00

Lesson 3: Many users feel their programming skills improve primarily due to practice, not their method

Many of the students said that using the Leap toolkit did not directly improve their programming skills. One student, when asked how their programming skills improved said, "I don't think it is because of Leap." But, when I asked them what is the most important thing they learned from the class they said "I have an idea how to estimate time, although not too accurately, but I can get a range." This knowledge came from Leap, not practice.

Many other students also reported that their programming skills improved primarily due to practicing programming and not because of the Leap toolkit. But, when asked what they learned from the class most said they learned how to estimate and plan their projects. It seems that many students do not consider project estimation a programming skill. Yet, many of them felt that the ability to plan their projects and manage their time was extremely important.

Lesson 4: To reduce measurement dysfunction, make the results less visible

The problem of measurement dysfunction in software process improvement is very important. It is even more important in a classroom setting. To teach students how to improve their own software development process without having the students fake their data is difficult.

In this class, none of the students reported sharing their Leap

data with other students. They did not compare their productivity or estimation abilities. In a previous class that I taught using the Leap toolkit, the students were actively comparing their productivity and estimating abilities. The major difference between the two classes was that in the previous class the students printed out their Leap data, while in the current course the students turned in their Leap data on floppy disks. With paper copies of their data the students could easily compare their process data and compete with each other. This competition may lead to measurement dysfunction.

Lesson 5: Partial defect collection and analysis is still useful

Many students felt that recording their important defects was very valuable. One student said, "I think collecting defects is good. You find out what mistakes you make." Another student said, "I still have a chance to go back look at them [defects] sometimes... So when I look at it ... the chance it will happen is kinda less." Students felt this way even though recording every single defect is not required for Reflective Software Engineering, unlike the PSP.

Lesson 6: Tool support should require few machine resources

Most of the students complained that the Leap toolkit was too slow and that it took up too much memory when it was running. Any tool supporting empirical software developer improvement should not adversely affect the developers environment. One student remarked "I think it's not very convenient" and that they would probably not use the Leap toolkit in the future.

Lesson 7: Experience may lead to overconfidence

Looking at the estimation accuracy data for the students in the class, I noticed that there was a difference between the estimation accuracy for students who reported more than five years of programming experience and students with less than five years experience. I divided the class into two groups, students who reported less than five years of programming experience and students with at least five years of programming experience. I compared their size and time estimation accuracies.

First, I looked at the students' size estimation accuracies. The experienced students, on average underestimated their program size by 32.5%. The less experienced students underestimated their program size by 8%. I found a significant difference between the two groups ($F = 11.3$, $P < 0.01$).

Second, I compared their time estimation accuracies. The experienced students had an average estimation error of -17%. The less experienced students had an average estimation error of -7%. Again, I found a significant difference between the two groups ($F = 3.94$, $P = 0.05$).

It appears that the more experienced students tended to underestimate the size and effort required for the programming assignment. This could be due to overconfidence in their programming skills and disbelieving the data that the Leap toolkit was giving them. By the eighth project, the experienced students were as accurate as the less experienced students.

4 Future Directions

Our current development effort focuses on providing a database backend for the Leap toolkit to support scaling of data storage. A single individual database repository will simplify data storage in the Leap toolkit.

Another major direction for this research is autonomous agents to help reduce the overhead for the user. These agents could help collect and analyze the user's data automatically and inform the user when they detect interesting patterns.

This research is supported in part by a grant from the National Science Foundation CCR 98-04010.

REFERENCES

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

[2] A. M. Disney. Data quality problems in the Personal Software Process. M.S. thesis, University of Hawaii, August 1998.

[3] A. M. Disney and P. M. Johnson. Investigating data quality problems in the PSP. In *Proceedings of the ACM SIGSOFT Sixth International Symposium on the Foundations of Software Engineering*, pages 143–152, Lake Buena Vista, FL, November 1998.

[4] 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.

[5] P. Ferguson, W. S. Humphrey, S. Khajenoori, S. Macke, and A. Matvya. Introducing the Personal Software Process: Three industry cases. *IEEE Computer*, 30(5):24–31, May 1997.

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

[7] B. Shostak. Adapting the Personal Software Process to industry. *Software Process Newsletter #5*, Winter 1996.