

## PROBLEM SOLVING SKILLS

Lynn Robert Carter<sup>1</sup>, William M. Waite<sup>2</sup>

**Abstract** — *This paper describes a three-year experiment to determine whether student performance can be improved by making their performance visible in quantitative ways and assisting them to reflect on deviations between desired and actual performance. The experiment focused on improving effort estimation for planning, improving effort allocation over the period of performance, and reducing the recurrence of previously reported defects in subsequent assignments. To accomplish this, a web-based tool was implemented and used to capture numerous student performance parameters as well as planning rationales, postmortem reflections, and feedback to the students.*

**Index Terms** — *performance improvement, quantitative study, reflection*

### INTRODUCTION

Any skill is taught by observing the student's performance of attempts to master that skill, evaluating the causes of poor performance relative to some standard, and suggesting ways to improve. In order to observe a student's approach to solving a problem, we need to make their performance visible in a measurable way. This is a very labor-intensive process, both for the student and the faculty member.

For the past three years, we have been experimenting with mechanisms to make problem solving behavior visible, to evaluate this behavior, and to modify it appropriately. The experiments have been carried out in one course, ECEN 4553, at the University of Colorado at Boulder. ECEN 4553 is a software development experience in which the students construct a compiler according to a standard decomposition.

Initially we were interested in the amount of effort required to accomplish each assignment and whether or not the students could leverage previous experiences to improve their effort estimation skills. We also saw a deadline-oriented pattern of behavior that concentrated considerable effort over the weekend when access to the professors and other resources were limited. Another troubling pattern was that defects reported to the student on one assignment would recur on subsequent assignments as though each assignment were an isolated event instead of part of an ongoing project.

We believed that by making various behaviors visible, measuring the behaviors in a quantitative way, and requiring the student to explicitly reflect on differences between planned and actual performance, we could improve their understanding of their work habits. We hoped that this

would lead them to experiment with behaviors that would improve their productivity.

While others have written about the improvement of performance in higher education, they tend to fall into two groups: academic performance improvement [3], and the study of performance improvement [1]. There is a paucity of published papers addressing quantitative studies of software development performance improvement within a higher education course or over a series of years of a course.

### THE COURSE

The fundamental characteristics of ECEN 4553 remained constant over the three-year experiment: Students work in teams and each team submits single copies of their designs and implementations. These team efforts account for 30% of the final grade in the course. To verify that each student has an appropriate understanding of the material, individual midterm and final examinations account for the remaining 70% of the grade. Individual students (not in teams) performed the first assignment and activities associated with our experiment.

As a field of study, compiler construction is well understood. Design patterns, formalisms, and tools exist for all of the major tasks except optimization. Using modern methods, nearly all of a typical compiler can be generated from appropriate specifications. Thus most of the design effort lies in problem recognition and solution selection.

Since there is so little novel work required, implementing a compiler is an excellent opportunity to focus students on the critical engineering issues of selecting the right tool for the problem at hand and integrating solution elements into a working system. The regular nature of the assignments also lends itself to addressing performance issues and issues of quality. (Why was your actual effort different from your plan? What might you do in the future to reduce this difference? At what point in your problem solving approach did each defect occur? What can you do to avoid similar defects in the future?)

While the basic course remained the same, the number of assignments, the work required for each assignment, and the approach used to measure and influence behaviors have changed over the three-year experiment.

In 1997, the course consisted of twelve distinct assignments implementing a subset of C for the MIPS. The data-gathering approach was strictly focused on estimating and recording effort.

<sup>1</sup> Lynn Robert Carter, Institute for Software Research International, Carnegie Mellon University, Pittsburgh, PA 15213-3890, lrc@sei.cmu.edu

<sup>2</sup> William M. Waite, Electrical and Computer Engineering, University of Colorado, Boulder, CO 80309, william.waite@colorado.edu

In 1998, the course was restructured into thirteen assignments of two kinds: “design” and “implement.” The problem was again to compile a subset of C for the MIPS. The data-gathering approach was expanded to capture both numeric data (expected effort for each activity and actual effort expended) and non-numeric data (rationale for each estimate and insight gained when actual effort figures were known). An attempt to capture defect data failed. (See section 1998, below.)

In 1999, the course was refined into eight assignments of which six were two-week design/implementation pairs. This time, the students were asked to implement an object-oriented language for the MIPS. The data-gathering approach was expanded yet again to include student-defined process steps for each phase of the assignment, effort scheduling, and more robust defect logging and reflection.

### THE EXPERIMENT

At the heart of the experiment is a plan/perform/reflect cycle. Each student creates a plan, performs the actual work, and reflects on what happened.

In 1997, students were asked to plan the amount of effort they would need to implement a fixed set of phases in an assignment's life cycle: plan, design, code, compile, test, and postmortem. During their implementation, they were asked to log the number of minutes they spent in each phase. In the postmortem phase, they were to reflect on the deviations between their plan and their actual effort. We developed a web-based tool, PEP (Personal Engineering Process), to support these activities. The tool eliminated the need for students to capture their plans and logs on paper and made it possible for the instructor and graders to easily review and comment on the students' work.

The amount of information being gathered has grown significantly over the three-year life of the experiment. In 1999, the students were asked to begin by explaining how they planned to go about solving the problem, the amount of effort they believed was required, and their rationale for this estimate. They then attacked the problem, recording the amount of effort that they actually used in the solution and any significant time-wasting defects they encountered along the way. Finally, they compared their planned approach with their actual performance and reflected on any differences. The professor and/or grader provided feedback on the plan and the reflection and logged defects found in the students' solutions to the problem. The students were required to submit a reflection on the source of their defects. The professor and/or grader provided feedback on the defect reflection.

PEP has also grown during the experiment. It now supports a “lessons learned” facility that automatically reminds student at the appropriate points in the planning process of lessons from previous assignments. Information can also be reviewed on a cumulative basis at any time.

The following sections provide a year-by-year survey of the experiment, our changing approach, our assessment of the effort, the resulting performance data, and the student feedback.

### The 1997 version of the experiment

Our experimentation began with the following hypothesis: Engineering performance (effort and quality) becomes more predictable when one takes the time to follow a regular process (solve similar problems the same way), uses historical data to build a performance baseline, and leverages this baseline in planning future work.

To verify this hypothesis, we added a text book [2] addressing individual time and quality management issues and made minor adjustments to the assignments in the existing ECEN 4553 class.

We used PEP to capture planned and actual effort data for each phase of a standard software life cycle [2].

Student data were secured via a username/password protocol so those students could see only their own work. Instructors could see the work of any student, and instructor access was secured via the same username/password protocol. Our attempt to capture defect plan and actual data failed due to defects in our first implementation of PEP. Students were required to turn in reports on paper, but the specific numbers or the accuracy of their estimates did not affect their grade. The PEP report was just a part of each assignment and no specific fraction of the assignment's grade was dedicated to PEP.

Implementation problems plagued our initial effort. The results shown in Table I do not indicate any improvement in ability to plan over the semester. At this point we believed that our implementation (both our overall approach and the specific PEP implementation) might have been at fault. One potential problem would occur when some students turning in well-considered plans and actual data earned the same mark as those turning in random information; another was the lack of any instructor/grader feedback. We elected to continue the experiment with changes to the class and the PEP implementation.

### 1997 details

The 1997 data was gathered from fifteen students in five groups over twelve distinct assignments, implementing a compiler for a subset of C on the MIPS.

PEP captured both planned and actual time spent. We attempted to count defects, but that part of the system didn't work and the effort was scrapped.

Table I

1997 effort in minutes and planning error (percent plan versus actual effort deviation of actual effort)

	Implementation Assignments								
	3	4	6	7	8	9	10	12	13
Effort	489	791	221	532	484	823	436	788	602
Planning error	46	55	86	21	34	42	53	36	38

**The 1998 version of the experiment**

In 1998, we wanted to see whether assigning specific credit to the PEP aspect of the assignment might improve the results, yet we did not want the students to believe that higher grades would be earned by smaller estimates, smaller actual efforts, or the accuracy of their estimates. To accomplish this, we elected to grade the quality of the reasoning behind their planning estimates and the quality of their analysis on deviations between their plan and their actual effort. This required the students to write a short justification of their estimates and to explain the insights they had gained by reflecting on the deviations between their planned and actual efforts. We provided detailed feedback to the students on these quality measures and how they might be made better.

We also rearranged the weekly assignments for the course, dividing them into two distinct classes: “concept-understanding” assignments and “implementation” assignments. A concept-understanding assignment usually involved answering a number of questions about the principles on which a certain compiler task was based, and perhaps a paper-and-pencil or implementation exercise to show the results of applying that task to a simple example. In the following implementation assignment, the students were required to modify their compiler to carry out that task. This rearrangement of the assignments did not change the underlying structure of the course.

The first two assignments dealt with general background information and the concept of program abstractions; they were concept understanding, but were not associated with any compiler task and therefore were not associated with implementation assignments. These were followed by five concept-understanding / concept-implementation pairs. According to this schedule, the concept-understanding assignment of the sixth pair fell on the week of the Thanksgiving holiday. That assignment was simply omitted.

PEP was reimplemented to address the system crashes and was dramatically expanded. Students no longer were required to submit paper copies of their PEP reports. The same username/password security protocols were employed to limit access and keep student data away from those without proper permissions.

Fewer implementation problems were present this second year, though one major PEP system crash related to

defect capture significantly reduced information for one assignment and served as an excuse for students to stop capturing defects for all remaining assignments. (Only four students from the class chose to capture defects using the system after the problem was corrected.) We believe the lack of clarity in how the capturing of defects and the reflection on them would benefit the student was also a factor. While the general trend in the planning estimation errors, as shown in Table II, was down (capturing rationales and insights was probably a good idea), the planning rationales showed little real insight about the use of production rates and size estimates to predict effort.

**1998 details**

The 1998 data was gathered from eleven students in four groups over thirteen assignments (seven design, six coding). The course project consisted of the same compiler from C subset to MIPS.

PEP captured effort estimate by phase, rationale for each estimate, defect estimate by phase, rationale for each estimate, actual effort by phase, actual defects by phase, student's insight after reflecting on deviations, grader's mark and comments on rationales and insights. (Very uneven student participation on the defect aspect of the experiment. Only four students logged defects.)

Table II

1998 effort in minutes and planning error (percent plan versus actual effort deviation of actual effort)

	Implementation Assignments					
	4	6	8	10	12	13
Effort	1027	300	739	494	820	790
Planning error	49	140	37	18	23	12

**The 1999 version of the experiment**

While the negative tone of the student feedback was a concern, the results gave us reason to continue the experiment. We still believed our initial hypothesis and now had some data to support our beliefs. Requiring planning rationales, reflection insights, and providing instructor/grader feedback appeared to be a significant benefit.

We now wanted to focus more of the students' attention on production rates and size estimates as the basis for planning, on scheduling their effort over the work week, and on the causes of defects in their solutions. Perhaps this additional focus might enhance the improvement in estimating skill over the semester, avoid the Sunday night crunch, and lead to a decrease in defects as well.

The course was again refined from a sequence of assignment pairs into a sequence of two-week assignments

consisting of two parts: 1) a design and a plan, 2) an implementation and two reflections. The underlying technical aspects of the course did not change, but the language to be compiled was changed from a subset of C to an object-oriented (OO) language that none of the students knew.

Once again, PEP was expanded. Students now were required to estimate explicitly the size of assignment deliverables and explicitly provide production rates for these deliverables as the starting point for their effort planning. Once they had estimated the effort for each deliverable, they then were required to spread that effort over the life cycle phases related to that deliverable. Each major life cycle phase was to be broken into a sequence of process steps as a form of checklist tailored for each student by that student. Effort figures calculated for each life cycle phase then could be adjusted to reflect unusual situations that might disrupt the flow of the work. Finally, the students were asked to schedule their life cycle phase effort over the three-week period allocated for each assignment. (Assignments were given in two-week intervals with the reflection on defects overlapping the first week of the next assignment.)

One interesting feature of the third PEP implementation was the addition of a "lessons learned" facility. During defect reflection, students were required to identify the root cause of each defect and the life cycle phase during which the defect was likely to have been injected into their work. The system then used this information to display indicators that there were lessons learned available at those places in the life cycle where defects were believed to have been injected. We hoped that easy access to their own lessons from previous assignments might help them to avoid some of the same mistakes on future assignments.

Only minor implementation problems occurred in the third year. The students, however, started complaining almost immediately about the complexity of PEP and how slow it was. A number of complaints indicated that students either did not read the on-line documentation or it was not helpful to them. While the magnitude of the error in planning started out quite large, it dropped regularly and quickly over the course. The additional complexity of PEP, the lack of experience in how to present the concepts on the web (both the documentation and the PEP interface), and concept overload (PEP concepts as well as compiler construction concepts) could be at the heart of the student complaints.

Had there been no serious complaints, we would have concluded that this third implementation was an improvement over the previous two implementations even though the size of the percentage deviations still indicated a lack of understanding and room for improvement in PEP, its documentation and the associated lectures. We did see a real change in students' deadline-oriented scheduling behavior over the period of the course (see Table IV). With the loud and uniform student complaints about time spent on PEP activity, however, it seems that the current PEP

implementation (both concepts and interface) is too complex to be used routinely.

Another important disadvantage of our 1999 strategy was the cost of grading the PEP materials and the lack of significance of the feedback to the students. It appears that many students failed to read the detailed grading comments, because they repeatedly lost points making the same mistake on assignment after assignment. The effort to provide these comments varied from a low of ten minutes to a high of thirty minutes per submission with an average over the semester of fifteen minutes. Since each student makes three PEP submissions (one plan and two reflections) per assignment, it took roughly 45 minutes to grade one assignment for one student. Over one hundred hours of effort were required during the semester to do just the PEP grading.

**1999 details**

The 1999 data was gathered from nineteen students in six groups over eight assignments, six of them two-week design/code pairs. The course project consisted of a compiler from an OO language to MIPS.

PEP captured time spent, defects, and forced detailed scheduling plans. PEP also captured reflection on actual vs. plan and defect insertion as well as grader feedback and the number of points awarded for each item.

Table III

1999 effort in minutes and planning error (percent plan versus actual effort deviation of actual effort)

	Implementation Assignments						
	1	2	3	4	5	6	7
Effort	228	315	676	789	1593	1263	836
Planning error	345	260	156	131	64	16	61

Table IV

Allocation of Effort Before and During the Weekend Before the Assignment is Due in the 1999 Experiment (percent of total assignment effort)

	Implementation Assignments						
	1	2	3	4	5	6	7
Before	55.4	39.6	41.6	44.4	62.3	70.5	52.3
During	40.8	52.7	43.8	49.0	33.7	24.1	43.1

(These figures do not add to 100 percent as any effort expended in the third week is not accounted for.) The final assignment is unusual in that there was no design component to the assignment and considerable time conflicts with unusual end-of-semester work from other courses. These issues may be a factor in the size of the planning error.

### CONCLUSIONS

Students can learn to improve their ability to estimate the effort needed to implement recurring tasks, but their insights into the process do not develop quickly and they do not seem to value this growing skill. Moreover, our 1999 data indicate that their deadline-oriented behaviors can be changed, even when they assert that they have no control over their schedules.

We had believed at the start of the 1999 cycle that the bulk of the improvement would be due to the understanding that students must gain from making their planning rationale explicit, reflecting on their plan versus actual deviations, and understanding the detailed feedback they were given. The fact that this aspect of the assignment was assigned a specific fraction of the students' grades should have served as motivation for them.

The 1999 data do not necessarily support this belief. While it is true that Table III shows a dramatic decrease in planning error over the semester, the error in the last assignment is comparable with the error at the beginning of the semester in 1997 and 1998. This last assignment may be an anomaly, however. It differed in structure from the others in that there was no design component of the assignment during the planning period, and a number of the students were taking another project course with an extremely heavy workload in the last two weeks of the semester. If this assignment were ignored, the final figure from 1999 would be better than any of the results from the 1997 data and similar to the final figures from 1998.

We attribute the extremely high initial planning errors shown in Table III to the complexity of the 1999 version of PEP. The students were clearly confused by the system at the beginning of the semester, and the decrease in planning error could be attributed to their increasing ability to cope with that complexity.

Non-numeric data gathered by PEP in 1999 also fails to indicate that the improvement was due to increased understanding. Most students' planning rationales and reflections were not particularly insightful and it was not clear that previous experiences were consciously influencing their thinking.

One student, after a disastrous performance on the midterm exam, focused considerable effort on the PEP activities because they were well defined, he knew how to earn high marks, and his results were completely under his control. The compiler implementation assignments, on the other hand, had new concepts in each assignment and a high mark required the whole team to perform well. Regardless of his motivation, his planning rationales and reflections contained excellent insights.

### REFERENCES

- [1] Collofello, James S., "Integrating Process improvement Practices into an Undergraduate Software Engineering Course", *FIE '98, 28<sup>th</sup> Annual Frontiers in Education Conference, Conference Proceedings*, Vol., No 3, 1999, pp 1298-1301.
- [2] Humphrey, Watts, *Introduction to the Personal Software Process*, Addison-Wesley, 1997.
- [3] Usó, J.-L., P. Mitic, L.J. Sucharow, "The educational modeling contributions to the software engineering teaching according to the experiences in the new computer schedule of courses at the University Jaume I", *Second International Conference on Software Engineering in Higher Education, SEHE 95*, 1997, pp 91-97.