

# Applying the Personal Software Process in CS1: An Experiment

Lily Hou and James Tomayko

School of Computer Science, Carnegie Mellon University  
5000 Forbes Avenue, Pittsburgh, Pennsylvania 15213

## Abstract

The authors conducted an experiment in applying components of the Personal Software Process<sup>sm</sup> (PSP) described in Humphrey[2,3] to a large group of CS1 students. Half of the students were taught selected PSP principles and the other half were asked only to keep track of total time spent on programming assignments. Results indicate that PSP is of value not only to software professionals involved in large projects, or to students in a software engineering school, but also to novices at the CS1 level, regardless of their background.

## Introduction

Software development is still a highly labor-intensive activity. Despite the proliferation of methods and tools, most software is actually written by individual engineers producing relatively small components. Therefore, any improvement in the efficiency or productivity of individuals will result in overall gains for projects and the industry as a whole.

Watts Humphrey managed a program in the Software Engineering Institute that created the Capability Maturity Model (CMM)<sup>sm</sup>, a basis for evaluating and improving the ability of organizations to build software. Understanding that the CMM did not address the quality issue at the lowest levels, he began to work on what essentially is process improvement for individuals. At first experimenting on himself, and then later on groups of students in the Master of Software Engineering Program at Carnegie Mellon University, he developed the Personal Software Process (PSP)<sup>sm</sup>, explained in detail in Humphrey[2].

The PSP addresses quality and efficiency in software development directly. PSP users experience improvements in their ability to estimate the size and time it will take to build a component. They also greatly reduce their defect rates. Data published in Humphrey[2] and gathered by the authors clearly show that there is much value in applying PSP to daily work.

Until recently, the PSP is taught almost exclusively to practitioners. Realizing that there is much potential in teaching some aspects of the PSP to freshmen in college, Humphrey worked with instructors at Embry-Riddle Aeronautical University in Daytona Beach to adapt the PSP to CS1. The results are in [1] and in a new book [3].

---

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee.

SIGSCE 98 Atlanta GA USA

Copyright 1998 0-89791-994-7/98/2..\$5.00

We noted that gathering data on applying PSP has been exclusively done on one group at a time, with no similar group for comparison. We decided to teach key aspects of the PSP to 65 freshmen at Carnegie Mellon University, with a control group of equal size not receiving any PSP instruction. This paper explores the results.

## Background

At Carnegie Mellon, there is a separate "flavor" of CS1 that is required of all Science, Business, and Psychology majors. The course is taught in C++, and covers the fundamentals of programming up to and including arrays of objects. This Spring, we had approximately 130 students, primarily freshmen and sophomores, enrolled in the course. The course requirements consist of eight programming assignments and a large programming project, in addition to quizzes, exams and a four hour on-line mastery exam. Due to the diversity of backgrounds and interests, the programming assignments are somewhat "generic" in terms of the problems. The project requirement works well for the following reasons:

- allows the students to define a problem in a domain of their interest, and take it from start to finish
- serves to give them experience in the design and implementation of a larger program
- provides experience in working within a team
- allows the students to see many different programming applications when the projects are presented.

Each project team consists of two students. In a few exceptional cases, projects are completed by individuals. Sample projects included data bases for jobs, graduate schools, financial portfolios, tutorials for the human cell, heart, memory based experiments, and games such as mazes. At least a third of the projects involved graphics.

The course is conducted in recitation sections of either 25 or 50 students at a time, meeting twice a week for 80 minutes each. This allows for lecture followed immediately by hands-on time on the computer. Based on the survey handed out on the first day, the following chart shows the background of the students:

	<u>PSP</u>	<u>Non-PSP</u>
No Prior Programming	81%	76%
1 Year of Basic	13%	8%
1 Year of Pascal	6%	16%

On the same survey, one of the questions is: "What are your fears about taking this course?" The most frequent answer to this is: "Having to spend too much time in the cluster/getting stuck on a bug for hours." Given the course re-

quirements and the background of the students, this is not a surprise. While we did not want to lower any of the course requirements, we wanted to look for ways in which we could:

1. Help the students to track how they were spending their time, and, in particular, their programming time.
2. Introduce some basic software engineering concepts.
3. Discover if complaints that this course required excessive time had any merit.

### The Experiment

Because the course schedule was already tight, it was not feasible to exercise all of the PSP components, nor use the Humphrey[3] text in its entirety. The experiment we conducted included the following:

1. Time and week logs
2. Keeping logs on the number of compile errors
3. Time in Phase, Code Review and Estimating
4. Gantt chart for the project

#### 1. Time Logs:

Like Hilburn's[1] class, the students were instructed to keep track of time spent on the following four activities:

Class	Attendance at recitation.
Study	Time spent doing reading assignments , and/or preparation for quizzes and exams.
Program	Time spent on programming assignments.
PSP	The time spent recording the time logs, and entering data into electronic Excel sheets for hand in.

The students were provided with paper and electronic Excel sheets to record times. All times were recorded in minutes. The data was handed in electronically, in the form of weekly logs which were condensed from their time logs.

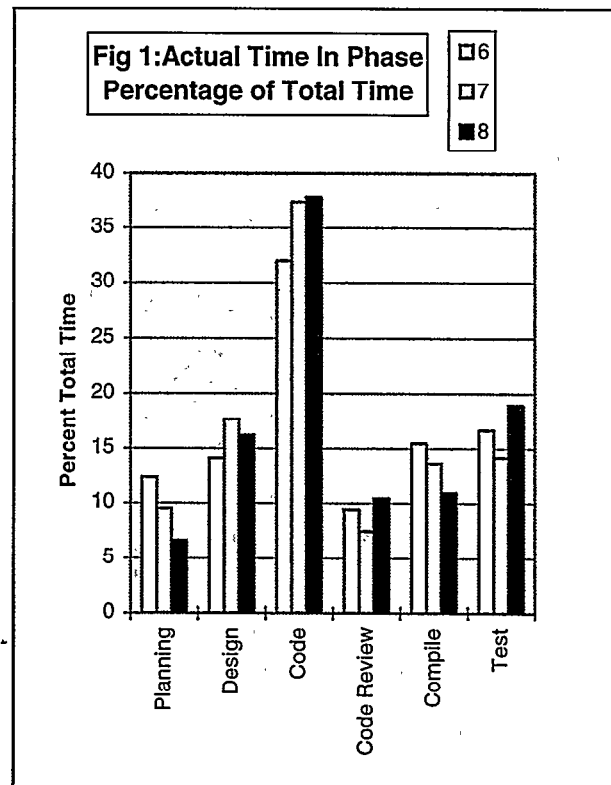
2. Error Logs and time in phase: Starting with assignment 5, they were asked to record the number of errors during the first compile, and time spent in the phases of compile, code, and test only.

3. Time in Phase, Code Review and Estimating: For assignment 6, we introduced code review. The students were provided with a list of errors to check for before they did their first compile. The time spent in each of the phases was also recorded. For assignments 7, 8 and, the project, the students were asked to estimate times for each of the phases before they began work.

4. Gantt Chart: Prior to beginning work on their project, the students were asked to use a Gantt chart to estimate the amount of time for each phase. These were estimates for each individual. Time spent as a team was not estimated.

### Data Analysis - PSP

First, we present data collected from the PSP group, including the average time spent on PSP and on the course.



### Time in Phase

Figure 1 shows the time in phase for assignments 6 through 8. Notice the consistency in the bulk of the time spent in coding, and in contrast, the time spent doing design. This is typical of many novices who equate coding with design. In addition there was no requirement to hand in any kind of design document, prior to the actual completed assignment.

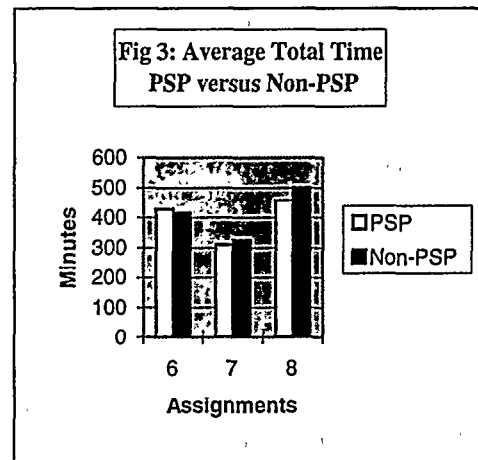
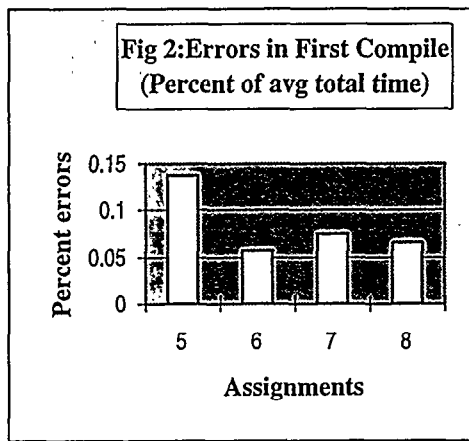
### Code Review

As stated earlier, the students started recording the number of errors in the first compile starting with assignment 5. Code review was introduced for assignment 6. We note two major benefits:

**Compile Time:** From Figure 1, we see how there is a steady drop in the amount of time spent on compile.

**Compile Errors:** The raw number of errors was weighted by the total time taken to complete each assignment. Figure 2 shows the huge improvement as a result of the code review. We focused on compile errors, because we believe that for a novice, it represents a "major hurdle" for any programming assignment.

The above results are even more significant, considering the fact that each assignment was progressively more challenging, both in terms of the programming constructs exercised, and the length of the programming assignment. For example, assignment 5 involved reading data from a file into a one dimensional array and doing basic computations. For assignment 6, they implemented the game of life, with a two dimensional array representation.



### Year to Date Average Time in the Course

The following year-to-date data was processed from the week logs:

Week Log#	Average Total Time in Minutes		
	Section 1	Section 2	Section 3
1	455	575	564
5	480	493	521
11	417	421	494

Week Log#	Average PSP Time in Minutes		
	Section 1	Section 2	Section 3
1	22.8	13.1	14.3
5	18.4	10.8	15
11	14.6	9.6	11.5

The first week log was actually week 3 of the semester, and so on. We did not collect any data in the first two weeks, because there is typically a lot of movement among the sections as the students try to work out their course schedule.

Initially, we did not even notice that Section 3 seemed to have the highest average total time. However, when the above data was presented to the students at the end of the semester, one of the students pointed out that it was probably because that section had exclusively freshmen. Upon examining our rosters, we found that, that was indeed the case. The reason the average PSP time was higher for Section 1 was because some students seemed to have a difficult time with Excel and handed in the wrong file several times. However, the overall average time is still 12 minutes/week.

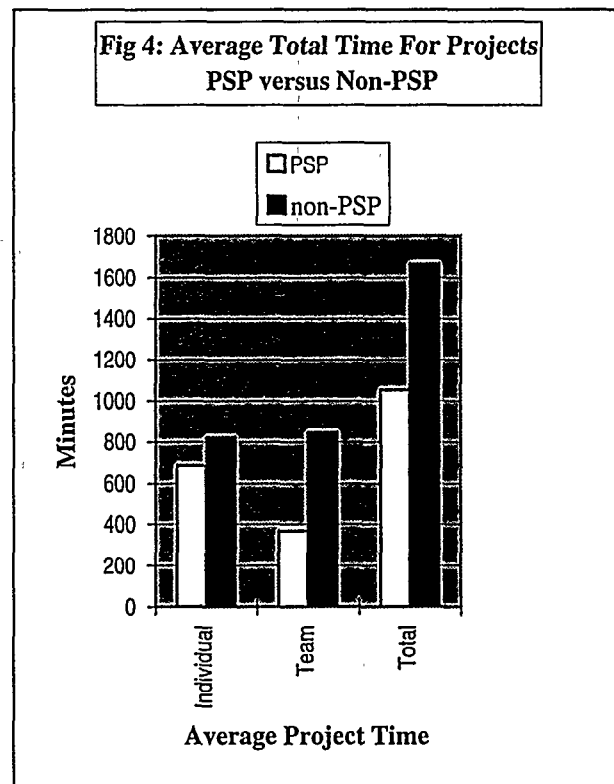
We do not present data on estimations, because we do not think the data collected was sufficient.

### Data Analysis - Comparison

Because the only common denominator for the PSP and non-PSP was total time spent on assignments, we present charts comparing assignments 6 - 8 (figure 3) and the project (figure 4). Data on code review time seems to suggest that the reason the PSP group took longer for assignment 6

may be because this was the first time they did code review. Notice that the differences are not significant.

However, the big "payoff" came with the project. Figure 4 shows the total time spent on the project. The students were asked to keep track of time spent individually, and as a team, when they integrated their respective parts. Notice the wide gap in the PSP and non-PSP groups. We believe this result was due, in large part, to the fact that the PSP group practiced code review prior to compiling, and tested their code, so that, when they got together as a team, there was much less time spent in debugging individual code. The bulk of the time was then taken up with integrating and testing the project as a whole. Another thing to note is that there were four incomplete projects handed in with the non-PSP group, versus one in the PSP group. We believe the Gantt chart was helpful in keeping them on track. We found this use of the Gantt chart to be very effective.



## Final Grades Comparison

According to Humphrey[2], PSP helps you improve the quality of your work. It does not necessarily increase your productivity, nor help you work faster. The following chart shows final grades for the two groups:

<u>Letter Grade</u>	<u>PSP</u>	<u>Non-PSP</u>
A	9%	9.5%
B	41%	42%
C	33%	33%
D	13%	8%
R	4%	8.5%

When we consider that there was a larger percentage of students in the PSP group with no prior programming experience, these results are significant.

Another conclusion from this data is that the talented students do well, regardless of how they structure the work time. However, the less talented students do better under PSP discipline than if left alone. Since the universe of software developers contains many of average talent, PSP can increase quality of work of a key segment of the industry.

## Lessons Learned

The time in phase data reveal how the students approach programming. Over years of teaching novices, we have found that it is difficult for novices to resist the urge to sit down in front of a machine and start typing code. Having some code that "runs" seems to give them a sense of accomplishment. For example, we have observed many students who will code the entire solution, without much structure to the program. When asked why they did that, they would reply, "Oh, as long as I have something running, it is not hard to clean it up." Sometimes that "clean-up" can take several hours. We would like to point out that both groups were taught the various phases in software development. However, the non-PSP group was not required to hand in anything regarding time spent in the various phases. Thus, it is up to the instructor to not only teach them a disciplined approach to programming, but also to ensure that they follow that discipline.

The weekly logs gave us a more concise picture of the amount of time students put into the course. While it is true that this is one of the more demanding courses in the curriculum, we are encouraged now by the evidence that the time spent is well within that allowed for the number of units that this course is worth. This data is useful not only to the instructor, but will also be made available to new students to help them plan their schedule.

At the end of the semester, the students in both groups were asked to fill out a survey. The PSP group were asked whether they considered the exercises to be helpful, and if so in what way. Those who said it was useful, said it helped them to schedule their time better, and many saw the value of doing code review. The students who did not find it useful, complained about the extra work they had to do. The other complaint was related to inadequate feedback on the data handed in. The non-PSP group was asked which phase took them the longest. The majority of them stated

that it was debugging. About 25 percent said they did everything together, i.e. they were unaware of time spent in each phase.

## Conclusions and Next Steps

As stated in the introduction, PSP has been used mostly by software professionals. The experiment by Hilburn[1] was on a group of students at a school that has a primary focus on Software Engineering. Based on our experience, we contend that there are components of PSP which are of extreme value to novice programmers, regardless of their background.

While the SAT scores and programming background of students may vary from school to school, the fundamental problem of time management is similar for freshmen and sophomores. The keeping of time logs is a step in the right direction to help them become aware of how they spend their time. As one of our freshmen put it, "I had to go find another cluster to do my work, because I did not like to record interruptions from my friends who kept stopping by."

Our results show not surprisingly that students learn what they are taught. In the absence of any disciplined method, they will choose to complete their program with minimal planning or design. They will also be more likely to graduate from the course with an extremely frustrated view of programming, i.e. spending long hours debugging.

Based on the data and feedback from students, we plan to continue the experiment this fall. This time, we will include all of the students. We are currently looking into what kinds of tools and exercises we can use to encourage students to do more design prior to coding. We plan to expand on the error logs, to include runtime errors. Work is also underway to develop more tools to process the data collected, so that we can give better and more timely feedback to the students.

## References

- [1] Hilburn, Thomas and Towhidnejad, Massood, "Doing Quality Work: The Role of Software Process Definition in the Computer Science Curriculum," In The Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education, February 1997, 277-281.
- [2] Humphrey, Watts S., "A Discipline for Software Engineering." Addison-Wesley, Reading, MA 1995.
- [3] Humphrey, Watts S., "An Introduction to the Personal Software Process," Addison-Wesley, Reading, MA 1997.