Philip M. Johnson and Anne M. Disney

The Personal Software Process: A Cautionary Case Study

In 1995, Watts Humphrey introduced the Personal Software Process in his book, *A Discipline for Software Engineering* (Addison Wesley Longman, Reading, Mass.). Programmers who use the PSP gather measurements related to their own work products and the process by which they were developed, then use these measures to drive changes to their development behavior. The PSP focuses on defect reduction and estimation improvement as the two primary goals of personal process improvement. Through individual collection and analysis of personal data, the PSP shows how individuals can implement empirically guided software process improvement.

The full PSP curriculum leads practitioners through a sequence of seven personal processes. The first and most simple PSP process, PSP0, requires that practitioners track time and defect data using a Time Recording Log and Defect Recording Log, then fill out a detailed Project Summary Report. Later processes become more complicated, introducing size and time estimation, scheduling, and quality management practices such as defect density prediction and cost-of-quality analyses.

After almost three years of teaching and using the PSP, we have experienced its educational benefits. As researchers, however, we have also uncovered evidence of certain limitations. We believe that awareness of these limitations can help improve appropriate adoption and evaluation of the method by industrial and academic practitioners.

PSP Benefits

Since the PSP's introduction, several case studies (M. Ramsey, "Experiences Teaching the Personal Software Process in Academia and Industry," *Proc. 1996 Software Eng. Process Group Conference*, 1996; Barry Shostak, "Adapting the Personal Software Process to Industry," *Software Process Newsletter,* No. 5, Winter 1996; Pat Ferguson et al., "Introducing the Personal Software Process: Three Industry Cases," *Computer*, May 1997, pp. 24-31) have reported positive experiences. For example, in one broad-ranging study, researchers from the Software Engineering Institute analyzed data submitted to them by 23 PSP instructors at both academic and industrial sites. The report concludes that the PSP improved performance in size estimation and effort estimation accuracy, product quality, process quality, and personal productivity, without any loss of overall productivity (Will Hayes and James 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 Eng. Inst., Pittsburgh, 1997).

At the University of Hawaii, we have taught and used the PSP for more than two years. We have also had positive experiences with it, and the results of our analyses mirror those reported by other researchers. For example, our final PSP project assignment typically requires that each student design and develop a 500- to 1,000-line Java program. Using the personal data they collect on their first eight projects, many students estimate both the size and time required for this final project with 95 percent accuracy. For students who typically enter the course believing that "software estimation" is an oxymoron, this experience proves revelatory. Some typical post-course evaluation comments include

♦ "I thought I was a good programmer, but after using PSP I realized that I was nothing back then."

EDITOR: Shari Lawrence Pfleeger • Systems/Software • s.pfleeger@ieee.org

♦ "I must admit, when I started this course, I understood what we were supposed to do in good software engineering, but I never really did it. Now I understand the reasons behind these practices and the benefits of actually following a process instead of just jumping right into coding."

The amount of PSP paperwork and manual calculation has long concerned us.

♦ "By executing and learning this process I know way more about software engineering than when I started this course."

We also practice what we preach...and teach. We applied PSP concepts to the development of a "Personal Thesis Process" for use by graduate students. One of us, Anne Disney, has consistently used PSP in her workplace for more than two years, and has recorded PSP data for more than 120 commercial database development projects—likely the largest collection of PSP data on a single developer's industrial practice in existence.

A Crushing Clerical Overhead

Although these results undeniably speak well of the PSP, we have long been concerned with the amount of paperwork and manual calculation involved in the original PSP curriculum. A software system developed using PSP2.0, for example, requires that students fill out 12 separate paper forms, including a project plan summary, time recording log, defect recording log, process improvement proposal, size estimation template, time estimation template, object categories worksheet, test report template, task planning template, schedule planning template, design checklist, and code checklist. These dozen forms typically yield more than 500 distinct values that each student calculates and records manually for a single project. If you think this sounds tedious, consider the fate of the instructor, who must check all these values. We estimated that in a recent PSP course, the instructor manually checked over 31,000 PSP data values for the nine projects developed by 10 students, in addition to checking the actual software projects themselves.

There is nothing wrong with a heavy workload when the end justifies the means. In the case of manual PSP, however, we began to wonder if the clerical overhead involved in completing and checking the forms by hand might significantly affect the ability of the PSP to drive process improvement.

Consider how the PSP actually works. Through a process of collection, the developer generates an initial empirical representation of her personal process. She then augments this initial representation with additional analyses, thus deriving the data that can give her the insights necessary for personal process improvement.

Ultimately, the goal of the PSP is to generate a model of the user's actual software development behavior accurate enough to support process improvement. In our initial experiences with manual PSP, we caught dozens upon dozens of clerical and other errors made by students during the course. If we were catching so many errors, we reasoned, could many more errors be slipping through? If so, were these errors distorting the behavioral model used to support process improvement?

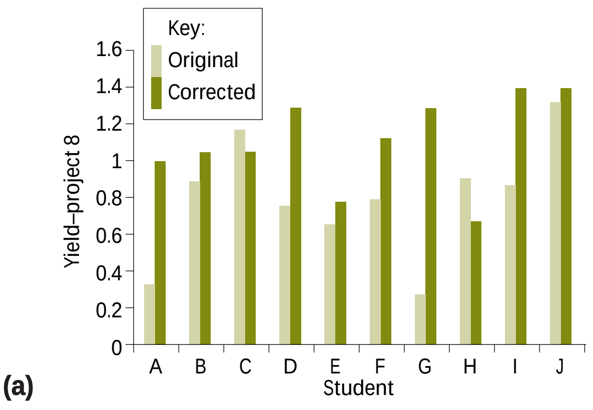
A Case Study To Check PSP Data Quality

To answer this question, we performed a case study as part of a class of 10 students learning the PSP in 1996. Our study design involved teaching the class using the manual PSP method, augmented with certain curriculum modifications designed to improve data quality. For example, we instituted technical reviews of the PSP data and generated supplemental forms to clarify the process of certain multistep estimation techniques. Next, we designed and implemented a database system that eventually stored more than 30,000 data values from the paper forms. The database system let us compare the student-generated empirical models of their behavior with the database-generated empirical models to determine if any differences existed.

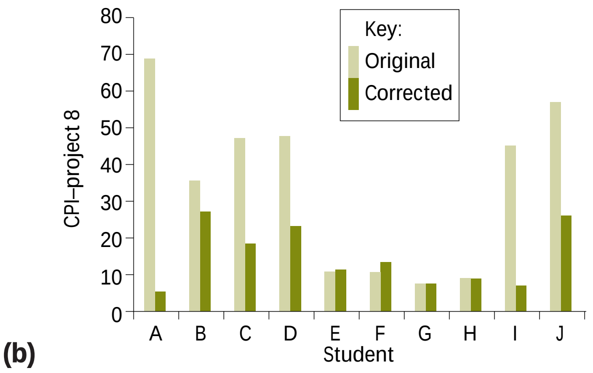
At the start of this case study, we anticipated that the database program might uncover 50 to 100 errors; to our astonishment, the program discovered 1,539. We also found that, in some cases, these errors were not simply noise in the model, but did appear to affect some of the important PSP measures, such as Yield (the percentage of defects discovered that were removed before the first compile) and the

Cost-Performance Index (a measure of how well the planned effort predicts actual effort). Figures 1a and 1b show the differences between the original values calculated by students and the "corrected" values calculated by our database program.

You could certainly generate alternative explanations for such a high number of errors. Perhaps these 1,539 errors were simply the result of poor instruction and/or project correction. As tempting as this explanation might be, the data does not appear to support it. The 1,539 errors represent only 4.8 percent of the total number of data values, which means that the instructor checked more than 31,000 data values by hand with over 95 percent accuracy. Although there is always room for improvement in any instructional context, 95 percent accuracy does not support the idea that the results are due to poor instruction.



(a)



(b)

Figure 1. Our case study sought to determine if the clerical overhead of completing the PSP's forms compromised its accuracy. For sample Project 8, the light green bars show students' original values for **(a)** Yield, the percentage of defects found and removed before the first compile, and **(b)** the Cost-Performance Index, a measure of how well the planned effort predicts actual effort. The dark green bars show the corrected values, derived from a database-generated empirical model. The database program uncovered 1,539 errors.

Recommendations

Do our results indicate that the PSP should be abandoned? Certainly not. We intend to continue using it for teaching, research, and our own industrial software development activities. Further, remember that our results are based upon data from a single course and that our design did not use a control group. Nevertheless, we recommend you consider the following points when deploying the PSP within your academic or industrial organization.

♦ The PSP can be used to teach useful software engineering skills, despite its potential for data quality problems. Some detractors claim that the improvements in product and process quality attributed to the PSP over the four months of the course would be achieved by any programmer who completed the 10 programming projects, regardless of whether that person used the PSP or not. We disagree emphatically. We have taught both introductory Java programming, which involved completing 10 or so Java programming assignments, and advanced software engineering, which involved completing a similar number of assignments while using the PSP. Although the results are not controlled, our anecdotal experience suggests that the PSP adds substantial value. While students in both contexts do improve with respect to syntax and programming idioms, only the PSP students acquire concrete software engineering skills that involve time and size estimation, defect-removal costs, design quality, and so forth. These skills, in turn, produce insights concerning process- and product-quality improvement not available to programmers who just hack code.

♦ Avoid teaching or adopting a manual, paper-based version of the PSP. Fortunately, since the time of our case study, automated tools for the PSP have become available. For example, East Tennessee State University makes a package called the PSP Design Studio, which is available online (Joel Henry, *Personal Software Process Studio*, http://www-cs.etsu.edu/softeng/psp/, 1997). In our opinion, it is vital that any PSP toolset you select replace the hard-copy forms from the original PSP curriculum with online equivalents, rather than simply compute values that must be transferred to the forms by hand. We say this because we provided the students in our study with tools that included spreadsheets and

Java-based code counting and estimation applets, but these unintegrated tools failed to prevent more than 1,500 errors.

♦ Avoid using PSP measures to evaluate the success of the PSP itself. It is tempting to use the changes in PSP measures that occur over the duration of the course as evidence of the method's success. This approach to evaluation is widespread in current PSP case studies, which frequently cite conclusions such as: "The improvement in average defect levels for engineers who complete the course is 58 percent for total defects per KLOC...." The problem is that such numbers are derived from the model of programmer behavior built by the PSP, which, as our case study shows, may not always accurately represent actual programmer behavior. Instead of using PSP measures to evaluate the PSP, case studies should use external, non-PSP measures. As an example of this approach, Pat Ferguson's report on industrial use of the PSP, cited earlier, found that acceptance-test defect density fell after the introduction of PSP into selected development groups.

♦ Avoid viewing automated support as a silver bullet for the problem of PSP data quality. Automated tool support has the potential to dramatically reduce or eliminate the analysis errors that occur while transforming a programmer's work records into analyses. Unfortunately, automated support can do little to guarantee that the programmer's recorded work accurately reflects her actual work. Thus, even if automated support is used to provide defect-free analysis, collection-stage errors could still lead to an inaccurate PSP model of programmer behavior.

For more details on this case study, including a discussion of the types of errors we found, their severity, and their origin, see either our technical report (Anne M. Disney and Philip M. Johnson, "Investigating Data Quality Problems in the PSP," *Proc. Sixth Int'l Symp. Foundations of Software Eng.*, *SIGSOFT '98*, 1998) or the thesis discussing this research (Anne M. Disney, *Data Quality Problems in the Personal Software Process*, master's thesis, University of Hawaii, Aug. 1998, http://csdl.ics.hawaii.edu/Research/PSP/PSP.html). ❖

Philip M. Johnson is an associate professor in the Department of Information and Computer Sciences at the University of Hawaii and director of the Collaborative Software Development Laboratory. He is a member of the IEEE Computer Society and ACM. Contact Johnson at johnson@hawaii.edu.

Anne Disney is a software engineer at Infoworld Management Systems in Malvern, Pennsylvania. This December, she will receive her MS in information and computer science from the University of Hawaii. Contact Disney at anne@ics.Hawaii.edu.