A Cautionary Case Study of the Personal Software Process

Philip M. Johnson
Anne M. Disney
Dept. of Information and Computer Sciences
University of Hawaii
Honolulu, HI 96822 USA
+1 808-956-6920
johnson@hawaii.edu

September 8, 1998

1 The Personal Approach to Software Process

The Personal Software Process (PSP) was introduced in 1995 in the book, "A Discipline for Software Engineering" [6]. This text describes a four month curriculum that teaches concepts in empirically-guided software process improvement. The text has been used as a basis for both university and academic courses to teach and improve software engineering skills.

Programmers using the PSP gather measurements related to their own work products and the process by which they were developed, and use these measures to drive changes to their development behavior. PSP focuses on defect reduction and estimation accuracy improvement as the two primary goals of personal process improvement. Through individual collection and analysis of personal data, the PSP illustrates how empirically-guided software process improvement can be implemented by individuals. The full PSP curriculum leads practitioners through a sequence of seven personal processes. The first and most simple PSP process, PSP0, requires practitioners to keep track of time and defect data using a "Time Recording Log" and "Defect Recording Log" similar to those illustrated in Figures 1 and 2, and then fill out a "Project Summary Report" similar to that illustrated in Figure 3. Later processes become more complicated, and introduce size and time estimation, scheduling, and quality management practices such as defect density predication and cost of quality analyses. Figure 4 is similar to one of the more advanced "Project Summary Forms" and illustrates aspects of the sophistication present in later processes.

2 Benefits of the PSP

Since its introduction, positive experiences with the PSP have been reported in several case studies [7, 8, 3]. For example, in one broad-ranging study, researchers from the Software Engineering

Table C16a Time Recording Log

Student	Jill Fonson	Date	1/2/98
Instructor	Philip Johnson	Program #	P1

Date	Start	Stop	Inter. Time	Delta Time	Phase	Comments
9/1	8:00	9:00	0	60	Plan	
9/1	2:00	3:00	20	40	Design	

Figure 1: A sample portion of a time recording log.

Table C18a Defect Recording Log

Name Jill Fonson **Program** 1

Date	No.	Type	Inj.	Rem.	Fix Time	Fix Def.	Description
9/2	1	50	Code	Com	1		Forgot import
9/3	2	20	Code	Com	1		Forgot ;
	3	80	Code	Com	1		Void in constructor

Figure 2: A sample portion of a defect recording log.

Institute analyzed data submitted to them by instructors of 23 PSP classes at both academic and industrial sites. The report concludes that the PSP improved performance in size and effort estimation accuracy, product quality, process quality, and personal productivity, without any loss on productivity [4].

At the University of Hawaii, we have taught and used the PSP for over two years. Our experiences with the PSP have also been positive, and the results of our analyses mirror those reported by other researchers. For example, our final PSP project assignment typically requires each student to design and develop a 500-1000 line Java program. Using the personal data they collect on their first eight projects, many students successfully estimate both the size and time required for this final project with at least 95% accuracy. For students who typically enter the course believing that "software estimation" is an oxymoron, this experience is revelatory. Some typical post-course evaluation comments are:

- "I thought I was a good programmer, but after using PSP I realized that I was nothing back then."
- "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."
- "By executing and learning this process I know way more about software engineering than when I started this course."

Table C14a PSP0 Project Plan Summary

Student	Jill Fonson	Date	9/1/98
Program	MeanStd	Program #	1
Instructor	Philip Johnson	Language	Java

Time in Phase (min.)	Plan	**Actual**	**To Date**	**To Date %**
Planning		60	60	14
Design		40	40	10
Code		110	110	26
Compile		60	60	14
Test		120	120	29
Postmortem		30	30	7
Total	90	420	420	100

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning	0	0	0
Design	1	1	9
Code	8	8	73
Compile	0	0	0
Test	2	2	18
Total Development	11	11	100

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning	0	0	0
Design	0	0	0
Code	0	0	0
Compile	8	8	73
Test	3	3	27
Total Development	11	11	100
After Development			

Figure 3: A sample portion of the PSP0 process summary form.

Table C55 PSP2 Project Plan Summary (Excerpt)

Student	Jill Fonson	Date	6/12/97
Program	P8	Program #	8
Instructor	P. Johnson	Language	Java

Summary	**Plan**	**Actual**	**To Date**
LOC/Hour	35	42	31
Planned Time	1445		13214
Actual Time		1525	15321
CPI(Cost-Performance Index)			.86
			(Planned/Actual)
% Reused	0	0	12
% New Reused	20	14	7
Test Defects/KLOC	4.7	5.3	8.2
Total Defects/KLOC	8.5	10.2	15.3
Yield %	45	48	47

Program Size (LOC):	**Plan**	**Actual**	**To Date**
Base(B)	235	235	
	(Measured)	(Measured)	
Deleted (D)	0	0	
	(Estimated)	(Counted)	
Modified (M)	0	0	
	(Estimated)	(Counted)	
Added (A)	467	673	
	(N-M)	(T-B+D-R)	
Reused (R)	0	0	
	(Estimated)	(Counted)	
Total New & Changed (N)	467	673	11343
	(Estimated)	(A+M)	
Total LOC (T)	702	908	15342
	(N+B-M-D+R)	(Measured)	
Total New Reused	0	0	2451
Upper Prediction Interval (70%)	502		
Lower Prediction Interval (70%)	902		

Defect Removal Efficiency	***Plan***	***Actual***	***To Date***
Defects/Hour - Design review	***3.2***	***1.5***	***3.2***
Defects/Hour - Code review	***4.5***	***2.3***	***4.5***
Defects/Hour - Compile	***12***	***14***	***12***
Defects/Hour - Test	***1.4***	***0.89***	***1.4***
DRL(DLDR/UT)	***2.3***	***1.7***	***2.3***
DRL(CodeReview/UT)	***3.2***	***1.7***	***3.2***
DRL(Compile/UT)	***8.6***	***15***	***8.6***

Figure 4: A sample portion of the PSP2 process summary form.

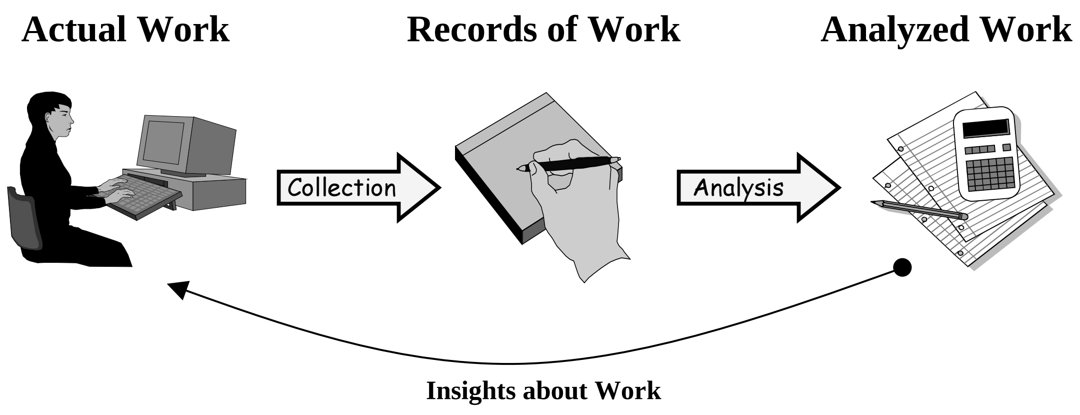

Figure 5: A simple model for PSP data quality.

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 over two years, and has recorded PSP data for over 120 commercial database development projects—likely the largest collection of PSP data on a single developer’s industrial practice in existance.

3 Issues with manual PSP

Although all of 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. For a software system developed using PSP2.0, for example, students fill out twelve 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 over 500 distinct values that each student calculates and fills in manually for a single project. If you think this sounds tedious, consider the fate of the instructor who has to check all of 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 ten students, in addition to checking the actual software projects themselves!

There is nothing wrong with a heavy workload, of course, when the means justify the ends. 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 impact upon the PSP itself? To better understand this issue, consider the simple “model of PSP data quality” in Figure 5. Through a process of *collection*, the developer generates an initial empirical representation (“Records of Work”) of her personal process (“Actual Work”). Through additional *analyses*, the developer augments her initial empirical representation with derived data (“Analyzed Work”) intended to enable process improvement through “Insights about Work”.

In other words, the goal of the PSP is to generate an empirical profile, or model, of the user’s actual software development behavior that is accurate enough to support process improvement. In our initial experiences with the manual PSP, we found we were catching dozens upon dozens of clerical and other errors made by students during the course. If we were catching so many errors,

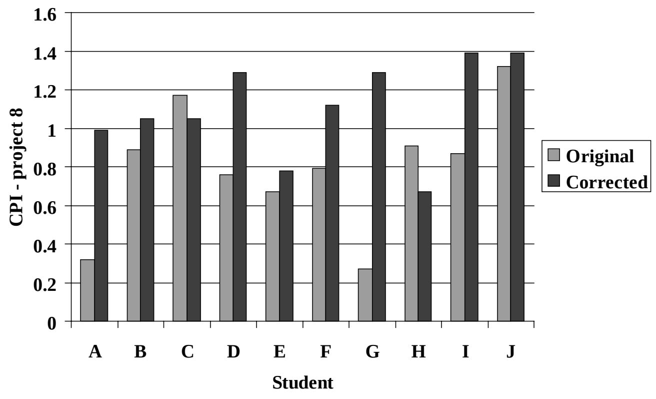

Figure 6: Effect of Correction on CPI

could many more be slipping through, and if so, were these errors making an impact upon the behavioral model used to support process improvement?

4 A Case Study to Check PSP Data Quality

To answer this question, we performed a case study as part of a class of ten students learning the PSP in 1996. Our case 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 multi-step estimation techniques. Next, we designed and implemented a database system that we used to enter each of the over 30,000 data values from the paper forms, and compare the results of each student's calculations with those computed by the database package. We used this to compare the student-generated empirical models of their behavior with the database-generated empirical models, and 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 1539 errors! We also found that in some cases, these errors were not simply "noise" in the model, but did appear to impact upon 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 6 and 7 illustrate the differences between the original values calculated by the students and the "corrected" values calculated by our database program.

One of the first questions we asked ourselves upon obtaining such a high number of errors was: Could this poor data quality be a simple result of poor instruction and/or project correction? As tempting as this explanation might be, the data does not appear to support it. The 1539 incorrect data values represent only 4.8% of the total number of data values, which means that the instructor checked over 31,000 data values by hand with over 95% accuracy. Although there is always room for improvement in any instructional context, 95% accuracy does not support the idea that the results are due to poor instruction.

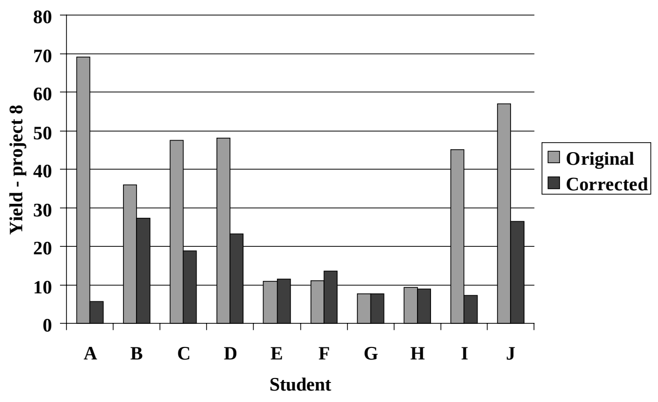

Figure 7: Effect of Correction on Yield

5 Recommendations

Do our results indicate that the PSP should be abandoned? Certainly not, and we intend to continue using it for teaching, research, and our own industrial software development activities. Furthermore, these results are based upon data from a single course, and a controlled experimental design was not used. Nevertheless, we recommend you consider the following when deploying the PSP within your academic or industrial organization:

1. *Despite the potential for data quality problems, the PSP still teaches useful skills.* Some detractors claim that the improvements in product and process quality attributed to the PSP over the four months of the course would occur in any programmer doing the ten programming projects, regardless of whether the PSP was usedor not. We disagree emphatically. We have taught both introductory Java programming (involving completion of 10 or so Java programming assignments) and advanced software engineering (involving completion of 10 or so Java programming assignments using the PSP). Although the results are not controlled, our anecdotal experience suggests that the 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 involving 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.

2. *Avoid teaching or adopting a manual, paper-based version of the PSP.* Fortunately, since the time of our case study, a few automated tools for the PSP have become available. For example, East Tennessee State University makes a package called the "PSP Design Studio" available on-line [5]. In our opinion, it is vital that any PSP toolset used must *replace* the hardcopy forms from the original PSP curriculum with on-line 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 including spreadsheets and Java-based code counting and estimation applets, but these (unintegrated) tools still did not prevent over 1500 errors from occurring.

3. *Avoid PSP measures when attempting to evaluate the success of PSP use.* Since the PSP produces measures of product and process quality, it is quite tempting to use changes in these

measures from the beginning of the course to the end of the course as evidence of the success of the method. 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% 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, one report on industrial use of PSP found that acceptance test defect density fell after the introduction of PSP into selected development groups [3].

4. *Automated support is not a silver bullet for the problem of PSP data quality.* As always in software engineering, automated tool support is not a panacea. Looking again at our model in Figure 5, automated support has the potential to dramatically reduce or eliminate the “analysis” errors that occur while transforming “Records of Work” into “Analyzed Work”. Unfortunately, automated support can do little to guarantee high quality “collection”—in other words, that the programmer’s “Records of Work” accurately reflect 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 [2] or the thesis discussing this research [1]

References

[1] Anne M. Disney. Data quality problems in the personal software process. M.S. thesis, University of Hawaii, August 1998.

[2] Anne M. Disney and Philip M. Johnson. Investigating data quality problems in the PSP. In *Sixth International Symposium on the Foundations of Software Engineering (SIGSOFT’98)*, Orlando, FL., November 1998.

[3] Pat Ferguson, Watts S. Humphrey, Soheil Khajenoori, Susan Macke, and Annette Matvya. Introducing the personal software process: Three industry cases. *IEEE Computer*, 30(5):24–31, May 1997.

[4] 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.

[5] Joel Henry. Personal software process studio. http://www-cs.etsu.edu/softeng/psp/, 1997.

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

[7] M. Ramsey. Experiences teaching the personal software process in academia and industry. In *Proceedings of the 1996 SEPG Conference*, 1996.

[8] B. Shostak. Adapting the personal software process to industry. *Software Process Newsletter No. 5*, Winter 1996.