Empirical Software Engineering, 4, 317–349 (1999)

A Critical Analysis of PSP Data Quality: Results from a Case Study

PHILIP M. JOHNSON johnson@hawaii.edu
Dept. of Information and Computer Sciences, University of Hawaii, Honolulu, HI 96822 USA

ANNE M. DISNEY adisney@ilhawaii.net
Dept. of Information and Computer Sciences, University of Hawaii, Honolulu, HI 96822 USA

Received November 18, 1996; Revised April 30, 1998

Abstract. The Personal Software Process (PSP) is used by software engineers to gather and analyze data about their work. Published studies typically use data collected using the PSP to draw quantitative conclusions about its impact upon programmer behavior and product quality. However, our experience using PSP led us to question the quality of data both during collection and its later analysis. We hypothesized that data quality problems can make a significant impact upon the value of PSP measures—significant enough to lead to incorrect conclusions regarding process improvement. To test this hypothesis, we built a tool to automate the PSP and then examined 89 projects completed by ten subjects using the PSP manually in an educational setting. We discovered 1539 primary errors and categorized them by type, subtype, severity, and age. To examine the collection problem we looked at the 90 errors that represented impossible combinations of data and at other less concrete anomalies in Time Recording Logs and Defect Recording Logs. To examine the analysis problem we developed a rule set, corrected the errors as far as possible, and compared the original and corrected data. We found significant differences for measures such as yield and the cost-performance ratio, confirming our hypothesis. Our results raise questions about the accuracy of manually collected and analyzed PSP data, indicate that integrated tool support may be required for high quality PSP data analysis, and suggest that external measures should be used when attempting to evaluate the impact of the PSP upon programmer behavior and product quality.

Keywords: Personal software process, defects, empirical software engineering, measurement dysfunction, automated process support

1. Introduction

> *The actual process is what you do, with all its omissions, mistakes, and oversights. The official process is what the book says you are supposed to do.* (Humphrey, 1995).

The Personal Software Process (PSP) was introduced in 1995 in the book, "A Discipline for Software Engineering" (Humphrey, 1995). This text describes a one-semester curriculum for advanced undergraduates or graduate students in computer science that teaches concepts in empirically-guided software process improvement. Since its introduction, experience with the PSP has been reported on in several case studies (Ceberio-Verghese, 1996; Ferguson et al., 1997; Humphrey, 1996; 1997). Although empirically-guided software process improvement is a key feature of other software engineering initiatives, such as the Capability Maturity Model (CMM) (Paulk et al., 1995), ISO-9000, and Inspection (Gilb and Graham, 1993), the PSP differs from these other approaches in important ways.

The CMM, ISO-9000, and Inspection discuss empirical software process improvement in the context of a large organization. Process improvement in this context requires the gathering and analysis of large amounts of data, within and across departments, generated by different people at different times. Indeed, inevitable personnel turnover means that the data collected from the working procedures of one set of people tend to generate measurements leading to process changes that affect the working procedures of a potentially different set of people. The substantial effort required to collect, interpret, and introduce organizational change based upon the measurements for a large organization leads to the need for an explicit software engineering process group (SEPG) whose mission is to manage empirically guided improvement. Although the utility of these approaches have been repeatedly validated, they leave the unfortunate impression that empirically-guided software process improvement is the sole province of large organizations who can dedicate teams of people to its enactment.

The PSP provides an alternative, complementary approach in which empirically guided software process improvement is "scaled down" to the level of an individual developer. In the PSP, individuals 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 provides a novel example of how empirically-guided software process improvement can be implemented by individuals regardless of the surrounding organizational context and the availability of institutional infrastructure support.

Since the PSP is a new technique, relatively little data exists on its use and effectiveness. Case studies typically report positive results, usually based upon the data collected during enactment of the PSP curriculum. One typical case study conclusion is that "during the course, productivity improvements average around 20% and product quality, as measured by defects, generally improves by five times or more" (Ferguson et al., 1997). Similarly, another study states that "the improvement in average defect levels for engineers who complete the course is 58.0 percent for total defects per KLOC and 71.9 percent for defects per KLOC found in test" (Humphrey, 1996). Indeed, our own PSP data yields similarly positive measurements for process and products.

In this paper, we report on a case study performed to assess the quality of PSP data—the measurements typically used to evaluate the effectiveness of the PSP as illustrated above. Our case study was motivated by our experiences teaching and using the PSP, which led us to suspect that the empirical measures gathered by the PSP may not, in all cases, reflect the true underlying process or products of development.

We hypothesized that problems with the quality of process data collected with the PSP could significantly change at least some of the measures produced by the PSP that are commonly used to evaluate its effectiveness. By "significantly", we mean something stronger than just a statistically significant difference between the recorded measurements and the actual underlying programmer behavior. We mean that the difference between recorded measures and actual behavior would be sufficient, at least in some cases, to lead developers to the wrong conclusion about how to improve their process.

To test this hypothesis, one of us taught a modified version of the PSP curriculum to a class of 10 students in the Fall of 1996. The course was augmented with features designed

to improve the data quality of the raw PSP data. We then entered the PSP measures into a database and subjected them to a variety of data quality analyses. These analyses uncovered over 1500 errors in the PSP data used by the students to track their work and motivate process improvements. Additional analysis yielded a seven part classification scheme for PSP data errors. Although we were not always able to generate corrected values for the data errors, partial correction lead to "significantly" different values for certain PSP measurements, confirming our hypothesis.

The remainder of the paper is organized as follows. Section 2 presents a description of related work, including an overview of the PSP itself, case studies of the PSP, and additions and enhancements to the basic approach. Section 3 presents a model of PSP data quality we devised to guide our investigation. Section 4 presents the case study; its design, instrumentation, data collection, analysis, and threats to internal and external validity. Section 5 presents the quantitative and qualitative results we obtained from the study. Section 6 presents our interpretation of these results. This section also introduces the concept of "measurement dysfunction", which is important to our interpretation of the results from this study and our recommendations for future research and practice.

2. Related Work

There is a small but growing number of research studies describing experiences with teaching and evaluating the PSP. This section overviews the relevant literature with special attention paid to three questions.

First, how does the research study evaluate the effectiveness of the PSP? The most common approach is to compare PSP data collected at the beginning of the course to data collected at the end of the course. We term these kinds of evaluations *internal measurement evaluation*, because measures collected using the PSP are used to evaluate the PSP itself. The less common approach is to compare some other measure of programmer/program quality collected before the introduction of PSP training to the same measure of programmer/program quality after the PSP training. We term these kinds of evaluation *external measurement evaluation*, because measures collected independently of the PSP training itself are used to evaluate PSP effectiveness.

Second, how does the research study verify the accuracy of the measurements used in evaluation–in other words, verify that the measurements actually reflect the underlying behavior of the PSP users? This question addresses one form of internal validity: that the research is designed in such a manner that the data can actually be used to answer the questions posed. The most common approach to measurement verification in PSP research is *manual inspection* of the data by the instructor. In some cases, the researcher employed *subject exclusion*, i.e. eliminating all the data from one or more subjects based upon data incompleteness or some qualitative appraisal of data quality.

Third, what issues related to data quality are raised by the research study? Does the research present any experiences related to low data quality and how they might be overcome? The most common improvement mentioned in the research is automation.

To begin, we present a brief overview of the PSP itself.

2.1. The Personal Software Process

In the PSP curriculum presented in "A Discipline for Software Engineering" (Humphrey, 1995), each student develops 10 small programs over the course of a semester using a sequence of seven increasingly sophisticated software development processes, labeled PSP0 to PSP3. For every program, the students record various measurements related to their personal development activities. Such measures include, for example, the time spent in each phase of development, the numbers of defects injected and removed during each phase, and the size of the resulting work product. Five analysis exercises focus on trends and relationships between all of the process data collected to that point in the course.

The initial programs use relatively simple PSP processes that establish a baseline set of process measures for time, size, and defects. Later programs employ more advanced PSP processes that extend these baseline process statistics. Although there are a myriad of extensions, most are of two general types: extensions to planning and extensions to defect management.

Extensions to the planning phase include estimates of the program's projected size, the projected time required to complete each of the phases, and the number of anticipated defects that will be injected and removed during development. The process by which these estimates are produced involves statistical analysis of historical correlations between designs (i.e. class and method counts) and actual size (in lines of code), between estimated size and actual time, between actual size and actual time, and between size and defects injected and removed. (While lines of code as a metric of size at the organizational level is almost uniformly condemned in the measurement literature, it seems to work surprisingly well in the PSP, since the measure is collected and applied to a single individual working in a single language in a relatively uniform domain.)

By the middle of the course, each student has typically recorded a hundred or more defects made during development. More advanced PSP processes implement defect management mechanisms to help students understand the impact of various kinds of defects and to drive process improvements intended to reduce future occurrence of important defect types. For example, since students record the phase in which each defect was injected and removed and the time required to fix it, it is possible to analyze the relationship between fix time and various characteristics of defects. One relationship nearly always present in student data is that the "longer" a defect is present, the more time it takes to remove it. Thus, defects injected during design and not removed until testing are nearly always more expensive to remove than, for example, defects injected during coding and removed during compiling. This outcome typically motivates students to put more effort and care into design activities. Later processes support such behaviors by providing active defect management mechanisms. For example, by analyzing defect data to determine the types of design defects made on prior projects, a student can generate a checklist to be used as part of a personal design review. This checklist can be used to ensure that when similar defects occur in future projects, they do not escape into code, compile, or test phases.

The most advanced PSP processes extend the basic PSP paradigm to support larger projects using a cyclic development method. In addition, PSP includes a meta-level process for

defining personal processes in non-software domains or for specific software organizational contexts.

Students record their PSP data onto one or more PSP forms provided in the textbook and made available electronically at the textbook publisher's website. Students fill out these forms manually and turn them in to the instructor. Supplemental PSP spreadsheets automate some of the calculations, which students manually transfer to the forms. The number of forms filled out increases from three for the first PSP process to over a dozen for the most advanced processes.

In this curriculum, PSP data quality is addressed in two ways. First, the instructor manually reviews all PSP data as it is submitted, and is instructed to accept only complete and accurate PSP data. If an error on a prior assignment is discovered, the student must go back and recompute both the measurements for the prior assignment and all assignments subsequently affected by this error. (The provided spreadsheets can ease this process.) Second, the instructor should exhort the students to approach this course professionally, and to recognize that only the completeness and accuracy of the data, not the actual values for defect density, productivity, etc. will be used in the determination of their grade for the course.

The chosen PSP programming assignments are also important to understanding issues of data quality, because the standard PSP assignments require students to build software systems that they later invoke to produce important PSP measurements and analyses. For example, program 2A builds a size counter, a tool necessary for obtaining one of the three primary measures in the PSP. Other assignments produce statistical calculations necessary for PSP analyses, including linear regression, correlation, multiple regression, and prediction intervals.

2.2. Case Studies of the PSP

The PSP text contains the original "case study" of the PSP. Throughout the book, Humphrey informally presents data collected from students in PSP classes at Carnegie Mellon University during pre-publication development of the text. Utilizing an internal measurement evaluation approach, Humphrey compares PSP data collected from early in the semester with data collected at the end of the semester, and finds trends toward decreased defect density, improved yield, and improved estimation accuracy. Data verification appears to consist of manual inspection in conjunction with some subject exclusion.

Humphrey provides more information about his experiences with the PSP in several related articles (Humphrey, 1994a; 1994b; Humphrey, 1995; Humphrey, 1996). In general, he presents results based on PSP data collected from over 100 engineers in both industrial and academic settings. Humphrey employs internal measurement evaluation and subject exclusion when engineers "reported either incomplete or obviously incorrect results." Results include an increase in size and time estimation accuracy, and a reduction in reported defects of approximately 50% over the course of the training.

In his master's thesis, Dellien describes his attempt to introduce a tailored version of the PSP into an existing industrial organization (Dellien, 1997). He analyzed the PSP, broke it down into components such as measurement and quality management, compared these com-

ponents with the existing processes used by the organization, and rebuilt a modified version of the PSP that addressed the perceived needs of developers but did not result in overlapping organizational processes. His evaluation concludes that using PSP in an industrial setting is different than using it in an academic setting due to differences in accountability, group context, lack of automation, and resistance to change. In his evaluation, Dellien found that it is difficult to objectively evaluate whether a PSP introduction has been successful or not, even when success is measured purely as cost-effectiveness.

Sherdil and Madhavji use the PSP as the basis for research on human-oriented improvement in the software process (Sherdil and Madhavji, 1996). This research attempts to measure an individual's "progress function", using such variables as productivity, defect rate, and estimation error. The analysis attempts to differentiate progress due to "first order learning" (i.e. simple task repetition, unrelated to the PSP) and progress due to "second order learning" (i.e. introduction of PSP techniques). Their evaluation uses internal measurement evaluation with standard PSP measures to track the progress function. Verification involved manual inspection of PSP data "... for consistency, accuracy and logical validity. Automatic tools were also used to verify the program size values. We also checked if two subjects were illegally exchanging code, but never found such an occurrence."

Hayes and Over report on a statistical analysis of a set of PSP data sets in an attempt to understand the overall impact of PSP education (Hayes and Over, 1997). This "case study on a set of case studies" involves 298 engineers who spent more than 15,000 hours writing over 300,000 LOC and removing about 22,000 defects, during the course of 23 separate PSP training programs in both academic and industrial settings. Hayes and Over used internal measurement evaluation to demonstrate improvement in size estimation, time estimation, and defect density, with no significant change in productivity. The report does not indicate that the authors performed any independent data verification or assessment of data quality, though the authors do claim that the data quality is "exceptional":

> Instructors enter the engineers' data into a spreadsheet provided with the course materials. The paper forms completed by the engineers are collected by the instructor, and the class data are analyzed and used to provide feedback to the engineers. During the training, trends in class data provide insights to the engineers, who may then compare their own data with that of the group. Given this careful focus on data and statistical analysis, the quality and accuracy of the data used in any given class tends to be exceptional.

Emam, Shostak, and Madhavji report on a study of the implementation of PSP in a commercial setting, with special emphasis on adoption success (Emam, Shostak and Madhavji, 1996). Unlike the majority of PSP studies, their evaluation focussed primarily on the long-term adoption success, rather than short-term changes in PSP measurements during training. In addition, they present several issues that may impact adversely upon PSP data quality and thus the use of internal measurement evaluation. For example, high levels of reuse can act as a confounding factor on trends in productivity and defect density. Trends in defect density could reflect changes in defect detection capabilities rather than changes in the underlying density of defects in the work product. Trends in yield could be primarily due to introduction of code reviews and not due to any other aspects of the PSP. Finally, they found that the paper intensive nature of PSP was problematic for professional engineers.

Ferguson, Humphrey, Khajenoori, Macke, and Matvya report on case studies of the PSP at three industry locations: Advanced Information Services, Motorola Paging Projects Group, and Union Switch and Signal (Ferguson et al., 1997). Unlike most other studies, PSP effectiveness was evaluated primarily using external measures. For example, post-development defect report data (either during acceptance test or field use) was used to compare the quality of PSP-developed projects with non-PSP-developed projects. Similarly, one of the studies compared schedule estimation error before and after PSP training. These case studies showed substantial improvement with respect to both defects and estimation on industry projects after introduction of the PSP. The use of external measures for evaluation, and the particular external measures chosen (such as customer-reported defects) eliminates the issue of PSP measurement accuracy. The researchers did not present any issues regarding data quality.

Claes Wohlin discusses the use of the Personal Software Process as a context for empirical experimentation (Wohlin, 1998). He finds that the PSP has several desirable aspects for experimentation, including a comprehensive specification of the experimental context (i.e. the PSP textbook (Humphrey, 1995)), relative ease in replication, and the ready availability of experimental measures. Among the challenges he cites are internal validity and external validity. In the case study used as an example, data from six students were removed because it was "regarded as invalid or at least questionable."

Our own case study, presented in this paper, is intended to contribute to this body of knowledge concerning the PSP by demonstrating the importance of explicit concern for data quality beyond what is covered in the PSP textbook. The next several sections present our case study and its results. In Section 6, we will revisit several of the case studies presented above and reinterpret them in the light of our findings.

3. Modeling PSP Data Quality

As we pursued this investigation of data quality problems in the PSP, we found it necessary to build a model and define some new terminology to clarify the approach of this research and its conclusions. This section presents the model and this terminology.

3.1. Collection and Analysis Stages

Figure 1 illustrates a simple two stage model of PSP data representing an iterative cycle of *collection* followed by *analysis*. The model begins with "Actual Work"—the actual developer efforts devoted to a software development project. As part of these efforts, the developer enters the collection stage during which she collects a set of *primary* measures of defects, time, and work product characteristics—the "Records of Work". Given these primary measures, the developer performs additional analyses during the Analysis Stage, many of which produce *derived* measures which are themselves inputs for further analyses. The secondary, derived measures and associated analyses are presented in various PSP forms—the "Analyzed Work"—and hopefully yield "Insights about Work" that change and improve future actual software development work activities.

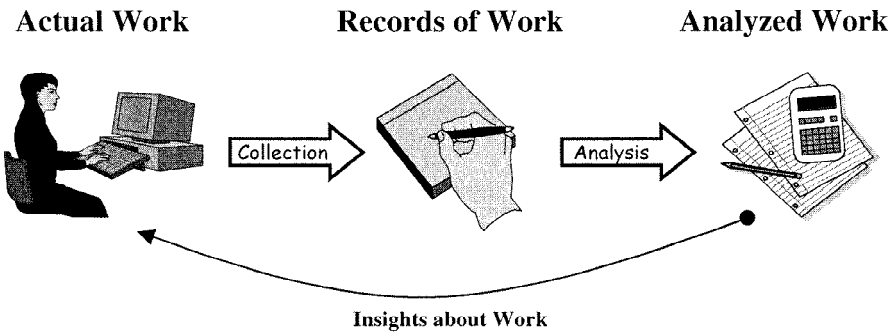


Figure 1. A simple model for PSP data quality. Through a process of *collection*, the developer generates an initial empirical representation ("Records of Work") of her personal process ("Actual Work"). Through additional *analyses*, the developer augments her initial empirical representation with derived data ("Analyzed Work") intended to enable process improvement through "Insights about Work".

3.2. Manual and Integrated Automation

We have also found it important to distinguish between the different levels of automation possible in the PSP. "A Discipline for Software Engineering" (Humphrey, 1995) presents an approach to PSP automation that we call *manual* PSP. Manual PSP refers to a situation in which PSP forms are filled out by hand, either by editing a copy of the form on-line, or by filling out a printed copy with pen or pencil. Manual PSP does not preclude the use of tools "on the side" to store historical data and to perform certain computations. In our classification scheme, if PSP tool support cannot eliminate manual manipulation and recalculation of derived measures, and thus guarantee their consistency and accuracy (assuming consistent and accurate primary measurements), then we view the level of automation as manual.

In contrast, we use *integrated* PSP to refer to the use of tools in which most or all of the derived measures during the analysis stage are calculated and placed into the forms automatically. Although integrated PSP should essentially automate all analysis stage calculations, there are limits to the ability of current technology to automate collection stage gathering of primary measures. For the foreseeable future, this part of the PSP will continue to be essentially "manual" in nature.

At the time we performed this case study, as was the case at the time of publication of "A Discipline for Software Engineering", there was no integrated software support for the PSP. Thus, the case study involved manual PSP, despite our extensive use of spreadsheets, program size counting tools, and statistical tools during the course. Since then, integrated support for the PSP has become available, including the Personal Software Process Studio tool produced by East Tennessee State University (Henry, 1997), and the Leap toolset at the University of Hawaii (Moore, 1998).

3.3. Omission, Addition, Calculation, and Transcription Errors

There are three basic ways to affect PSP data quality in the collection stage: errors of omission, errors of addition, and errors of transcription. Errors of omission occur when the developer does not record a primary measure related to defects, time, or the work product itself. If a defect occurring during "Actual Work" does not appear in the "Records of Work", then, for example, the PSP model of that work product's defect density will be lower than its actual defect density. If time spent on the work product is not recorded, then the PSP model of that developer's productivity will be higher than her actual productivity. Errors of addition occur when the developer augments the "Records of Work" with data not reflecting actual practice. For example, a developer, having made an error of omission to the point of having no time or defect data, may recover by simply inventing enough time and defect entries to make his or her PSP data appear plausible. Finally, errors of transcription occur when the developer does intend to accurately record his "Actual Work" in the "Records of Work" but makes a mistake during this process.

The presence of collection stage data quality problems is typically difficult to ascertain and difficult or impossible to rectify. In the PSP, primary data collection often feels both time consuming and psychologically disruptive. Many students complain that stopping to record defects disrupts their "flow" state, and that the time spent recording a defect—particularly for compilation stage errors—often exceeds the time spent correcting the defect. The PSP requires users to learn to constantly interleave "doing work" with "recording the work you are doing".

There are also three basic ways to affect PSP data quality in the analysis stage of manual PSP: errors of omission, errors of calculation, and errors of transcription. Errors of omission occur when the developer does not perform a required analysis of the primary data. Errors of calculation occur when the developer attempts to perform an analysis but does so incorrectly. For example, a developer might use a regression-based estimation method when the historical data is so uncorrelated that this method is invalid. Finally, errors of transcription occur when the developer makes a clerical error when moving data from one form to another.

Analysis stage data quality problems are typically much easier to ascertain and correct than collection stage data quality problems, *provided that the problems did not originate during the collection stage*. In other words, if one assumes that the work records accurately reflect the underlying work, then appropriate use of automated tools can reduce or eliminate analysis errors of omission, calculation, and transcription. On the other hand, since the quality of these analyses are totally dependent upon the quality of the work records produced in the collection phase, overall PSP data quality could be quite low even if the analysis stage is totally automated to eliminate all of its potential data quality errors.

4. The Case Study

To gain insight into the occurrence and significance of collection and analysis data quality problems, we conducted a case study. The case study was designed to investigate the

following hypothesis:

> *Data quality problems during collection and analysis can distort the PSP data's representation of the programmer's actual behavior, leading to invalid process improvement changes.*

4.1. Case Study Design

The case study began by teaching a one semester course on the PSP, modified in certain ways in an attempt to improve the quality of PSP data. The 10 students in the course submitted all of their paper forms to the instructor after each assignment. Some errors required students to correct and resubmit prior forms. The set of paper PSP forms collected over the course of the semester comprises the *original* PSP dataset, and can be thought of as one experimental treatment.

Next, the second author entered each data value from the original PSP dataset into a database system that she developed. This database system implements automated calculation of the derived measures, and detects a subset of the possible errors that can exist in a PSP dataset.

She also developed a second system to support correction of some of the errors found through the first database system. This system corrects a subset of erroneous data values according to a set of correction rules (specified in Section 4.5.1). Application of these rules produced the second, *corrected* PSP dataset, which can be thought of as the second experimental treatment. Note that we do not claim that this second dataset is completely correct, merely that it corrects a set of clearly inaccurate calculations from the first dataset.

Given this approach, the case study design is similar to a within-subjects comparison of a "control" treatment (the original PSP dataset) to the "experimental" treatment (the corrected PSP dataset). Our data analysis is designed to determine whether significant differences exist between these two treatments.

In addition to the test of our primary hypothesis, we used the database systems to perform several additional analyses on the observed errors to understand their cause and potential significance to PSP data values and the method itself.

4.2. The Modified PSP Curriculum

The projects used for this study were obtained from a software engineering class taught by Philip Johnson, in which the PSP was taught over the course of a semester using nine project assignments. There were ten students in the class, and 89 completed projects.

Because of the concern with data quality from prior experience teaching PSP, the instructor made four principal modifications to the standard PSP curriculum: increased process repetition, increased process description, technical reviews, and tool support. For replication purposes, a more detailed description of the curriculum used in this couse is available at the website: http://www.ics.hawaii.edu/∼johnson/613s98/.

Increased Process Repetition

In the standard PSP curriculum, students are assigned 10 programs during the semester (in addition to several midterm and final reports). Over the course of these ten programs, students practice seven different PSP processes, which means that the development process used by the students changes for seven out of ten programs. From our initial experience with the PSP, we found that the overhead of this almost constant "process acquisition" led to data errors and could overwhelm the effort related to actual development. To ameliorate this situation, the modified curriculum included only five PSP processes, enabling students to practice most processes at least twice before moving on to a new one. The modified curriculum also included only nine programs instead of ten, providing additional time in each program for data collection and analysis.

Increased Process Description

In our initial experiences teaching the PSP, the instructor found that students had a great deal of trouble learning to do size and time estimation correctly. For example, PSP time estimation requires choosing between three alternative methods for estimation depending upon the types of correlations that exist in the historical process data from prior programs. To help resolve this and other problems, the instructor added four additional worksheets: (1) a Time Estimating Worksheet to provide a guide through the various methods of time estimation; (2) a Conceptual Design Worksheet to help in developing class names, method names, method parameters, and method return values; (3) an Object Size Category Worksheet to help in size estimation; and (4) a Size Estimating Template Appendix to provide a place to record planned and actual size for prior projects.

Technical Reviews

At the completion of each project, students divided into pairs and carried out a technical review of each other's work. A two-page checklist facilitated this process. It included such questions as "Did the author follow the PSP Development Phases correctly?" and "Is the Projected LOC calculated correctly?" A second "Technical Review Defect Recording Log" form included columns for number, document, severity, location, and description. Students were given approximately 60 minutes to do the review. The technical review forms were submitted with the completed projects. The instructor reviewed the projects a second time for grading purposes, using the Technical Review Defect Recording Log to record any additional mistakes.

Tool Support

Finally, the instructor provided four spreadsheets to support records of planned and actual data values. In addition, students were provided with well-tested tools to count non-

comment source lines of code for Java programs, to compare two versions of a Java program and report non-comment lines of code added and deleted, and to perform certain statistical analyses. (In the textbook PSP curriculum, students "bootstrap" their environment by implementing these tools themselves. While elegant pedagogically, this approach unfortunately introduces a potentially significant source of data quality problems, since these freshly developed tools with no usage history are used to generate many of the measures used in later data analysis.)

In addition to these curriculum modifications, the instructor emphasized data quality throughout the course, as recommended in the textbook. For example, he augmented the lecture notes in the Instructor's Guide with fully worked out examples of the PSP process data for a fictitious student to show how data is collected and analyzed for each assignment and accumulated over the course of the semester. He dedicated lectures to collection and analysis of data periodically throughout the semester. He regularly showed the class aggregate statistics on class performance. He met with students individually and in groups throughout the semester to go over their assignments and PSP data while they were in the midst of planning, design, code, compile, test, and/or postmortem; but prior to project turn-in. He uncovered and removed many, many PSP data errors through these meetings which are not counted in our results. He did technical reviews of every assignment's PSP data, and circulated problem reports throughout the semester summarizing issues discovered from student data.

4.3. Case Study Instrumentation

We developed a database application to support analysis of PSP data from PSP0 to PSP2, using the Progress 4GL/RDBMS. In order to reduce opportunities for making mistakes, this tool was designed to require a minimum amount of user input and to provide the user with default values whenever possible. Apart from task and scheduling template values, the application automated all analysis stage calculations, from determining delta times for Time Recording Log entries to performing linear regression for size estimation. In addition, the application guides the user through the appropriate forms and fields in the order most appropriate for the current process and phase.

4.4. Data Collection

Once the database application was ready, we entered data from the student project PSP forms and compared each student value with the value computed by the application. Although every discrepancy between the manually generated data and the application-generated data could be considered an error, we only counted an error at its insertion point. For example, in a Time Recording Log entry for the Design phase, if *Stop* is incorrectly subtracted from *Start*, *Delta Time* will be incorrect. Even if all other calculations are done correctly for the rest of the project, *Time in Phase, Design, Actual*; *Time in Phase, Total, Actual*; *Time in Phase, Design, To Date*; *Time in Phase, Total, To Date*; *Time in Phase, To Date %*; and *Time in Phase, To Date* values for an indefinite number of future projects will

all be inaccurate to some degree. And this is just for the most simple process, PSP0! In more advanced processes, *LOC/Hour*, time estimation, *Cost-Performance Index*, and *Defect Removal Efficiency* values could all be affected for both the current project and future projects. To eliminate this combinatorial explosion in the number of errors, we counted this as a single error in *Delta Time*.

Although we analyzed the project data quite carefully, we do not feel confident that we have uncovered all or even most of the errors in this case study. While our database application does enable us to determine the correctness or incorrectness of values generated during the analysis stage of our data quality model, it provides only limited insight into collection stage errors. For example, in the Time Recording Log, it was possible to check the *Delta Time* computation, but not the accuracy of *Date*, *Start*, *Stop*, or *Interruption Time*. Of course, the tool could not, in general, detect the absence of entries for work that was done but not recorded. Two other areas that created similar problems were the Defect Recording Log and the measured and counted *Program Size* fields for the Project Plan Summary.

4.5. Data Analysis

In order to analyze the 1539 errors uncovered by the PSP data entry tool, we developed a second database application, the PSP Error Data Analysis Tool. For each error discovered, we tracked the person who made the error, the method by which the error was found (technical review, instructor review, or comparison with the PSP tool results), the assignment in which the error occurred, the PSP process used for that assignment, the PSP phase in which the student was working when the error occurred, the general error type, the specific error type, the severity of the error, the age of the error (number of assignments since the introduction of the PSP operation in which the error occurred), the incorrect and correct values (where applicable), and an optional comment for noting issues of interest in that error.

4.5.1. Error Correction

Although our initial analysis of our case study data revealed many errors, the sheer presence of errors might only lead to imprecision, rather than inaccuracy. In other words, it was possible that these errors were only "noise", similar in magnitude to naturally occurring random fluctuations in behavior, but not sufficient to actually change the trends or interpretations of PSP data.

To test this hypothesis, we attempted, where possible, to fix errors so that original and corrected versions of the data could be compared. It soon became clear that errors fell into three classes. First, there were errors where the correct value could be determined. This class included such values as *LOC/Hour* that were wrong simply because of an incorrect calculation. These errors were easily fixed by correctly performing the calculation in question. Second, there were errors where the correct value could not be determined, such as a blank *Phase Injected* for a Defect Recording Log entry. Fortunately, most errors in this class occurred in fields that didn't affect other fields, such as missing header data or missing

dates in the Defect Recording Log. Third, there were errors where the correct value could be guessed. In a Time Recording Log entry with *Start* 10:00, *Stop* 10:30, *Interruption Time* 0, and *Delta Time* 40; it is clear that there is a problem, but not clear which field is incorrect and should be corrected. However we can guess that there was a problem calculating *Delta Time* and assume that the other values are valid. To correct this third class of errors in an explicit and consistent fashion, we developed a set of rules. Underlying each of our rules is the assumption that primary data is more likely to be accurate than calculations performed upon it. The following lists each of the rules along with the number of times it was used in the case study.

Rule 1 (used 53 times): Defects in Time Recording Log entries should be handled by assuming that the start/stop/interruption times are correct and that the delta time is wrong, unless two Time Recording Log entries overlap. In that case, the preceding and following entries and the delta time for the current entry should be used to formulate plausible start/stop times. Generally this will mean starting the second entry where the first one stops.

Rule 2 (used 5 times): If a Time Recording Log is missing an entry for an entire phase, but the Project Plan Summary form contains a value for the phase under *Time in Phase (min.), Actual*, an appropriate Time Recording Log entry should be formulated with fabricated date and time values.

Rule 3 (used 28 times): For conflicts between a Defect Recording Log and a Project Plan Summary it should be assumed that the number of defects and the phases recorded in the Defect Recording Log are correct and that the discrepancy occurred as a result of incorrectly adding up the numbers of defects injected/fixed per phase and/or incorrectly transferring these totals to the Project Plan Summary form.

Rule 4 (used 10 times): If, for the Defect Recording Log, the total of all fix times for defects removed in a certain phase is more than the time recorded for that phase in the Time Recording Log, a Time Recording Log entry should be inserted with start and stop times that, combined with the existing Time Recording Log entries for the phase, will produce a delta time of the total fix times plus one minute for each defect. This will represent the minimum amount of time required to find and remove the recorded defects.

Rule 5 (used 1 time): To provide a value for a blank *Time in Phase (min.), Plan* field on the Project Plan Summary form, the value for *Time in Phase (min.), Actual* for the same phase should be used. Note that this rule, if used widely, would itself introduce error into the correction process. However, we used it only once on one project and it has negligible impact upon our results.

Rule 6 (used 62 times): Conflicts in *Program Size (LOC)* fields on the Project Plan Summary form should be handled by assuming that *Base, Deleted, Modified, Added, and Reused* are correct and that errors are the result of incorrect calculations for *Total New and Changed* and *Total LOC*. Actually, this is not a truly satisfactory assumption because *Total LOC, Actual* should be a measurement rather than a calculation and should therefore be relied upon. However, given correct values for *Base, Deleted, Modified, Added*, and *Reused*, it is possible to calculate *Total LOC*, whereas it is impossible to even guess at the correct values for the other fields. Unfortunately, defects in the *Program Size (LOC)* fields were some of the most common defects.

4.5.2. Data Comparison

After we partially corrected the project data according to the rule set, we investigated which values to compare to best reveal the effects of errors. Projects 8 and 9 had the most fields to compare since they were completed using PSP2, and provided the best opportunities for observing the cumulative effect of errors made in earlier projects. Project 9 was the best project for comparison because students had had the most practice in PSP by the time this project was completed and because it provided more time for cumulative effects to exhibit their true characteristics. Unfortunately one student did not complete this project, resulting in fewer data points for the final project.

One of the more interesting areas for comparison would have been size and time estimation. This was not possible due to the difficulties in adequately correcting the *Program Size (LOC)* fields. Instead, we selected a few fields from each of the other major sections of the Project Plan Summary, including some fields that resulted from fairly simple calculations but represented to date values from all nine projects, and other fields that were more local to the current project but were the result of more difficult operations.

4.6. Threats

We tested the hypothesis investigated in this study by comparing two PSP datasets: an uncorrected PSP dataset obtained from our students, and a (partially) corrected PSP dataset produced through automated analysis and implementation of correction rules. In this section, we discuss threats to the internal validity (whether the approach used is actually valid for testing the hypothesis) and to external validity (whether the results obtained in this study are applicable to external industry and academic practice of the PSP).

4.6.1. Internal Validity

One threat to internal validity is an instrumentation effect. This could have occurred in two ways. First, there could exist defects in the design and/or implementation of the database system used to create the partially corrected PSP dataset. To minimize this threat, great care was taken in the development of this database system to ensure the accuracy of its computations, and all data entered was re-checked at least once to ensure that there were no transcription errors. It is also relevant to note that Anne Disney, who designed and implemented the database, has worked professionally for many years doing database development using the same DBMS employed in this study.

A second threat to internal validity occurs from our use of correction rules. It is conceivable that a correction rule, if improperly designed, could introduce a systematic bias into corrected dataset that produces an artificial difference between the two datasets not related to underlying programmer behavior. To minimize this threat, we evaluated each of our rules for the potential presence of such systematic bias. One rule, in fact, does have the potential to produce this problem, but we used this rule in only one case in the entire dataset and our results are not sensitive to the specific value chosen.

4.6.2. External Validity

One threat to external validity is the sample size and nature of our subjects. Our sample size of 10 students leaves open the possibility that the results could be an artifact of the individuals involved in the study. A related threat involves the use of students for the study. Perhaps professional software engineers would approach the learning of the PSP in a different manner than students, given that the rewards and motivation structure in industry are quite different from academia.

Another threat to external validity is the instructor. Clearly, the level of PSP data quality during both collection and analysis is influenced by the quality of instruction. It may be possible to obtain different outcomes merely through alternative approaches to instruction. Furthermore, the instructor in this study has not attended the official SEI-sponsored PSP instructor training course. However, as discussed in Section 5.1, the PSP datasets submitted by students in the case study show precisely the same sorts of trends reported by other instructors, and as discussed in Section 5.2.8, the number of PSP dataset errors detected in our study is actually quite low, when viewed as a percentage of the total number of possible errors. In addition, the case study semester was the second time the instructor taught the PSP curriculum, and student evaluations were overwhelming positive. The case study outcomes do not appear to be the result of lack of instructor familiarity with the material or the result of simple student apathy regarding the course.

While we attempted to minimize these threats to both internal and external validity of this study, they are still real. The most effective way to evaluate these threats is through replication of this study in other environments using different data verification mechanisms, different subjects, and different instructors. We hope that this study will demonstrate the need and importance of such replication efforts in the PSP community.

5. Results

This section presents two types of results from the case study. First, we present the educational results, indicating that students did acquire substantial insight into software engineering during the semester and viewed the course as valuable. Second, we present the data quality results, obtained from a comparison of the original PSP dataset with the corrected PSP dataset according to our experimental design as discussed in Section 4.1.

5.1. Educational Results

Despite the discovery of data quality problems to be reported below, we still view the case study semester as an unqualified success from an educational standpoint. From a quantitative perspective, student data for the course parallels the positive outcomes from other PSP case studies:

- Average defect density showed a downward trend from around 200 defects/KLOC to around 50 defects/KLOC, a 75% decrease.

- Average productivity showed a very slight positive trend, from around 15 LOC/hour to around 20 LOC/hour.

- Time and size estimation showed dramatic improvement. On the last program, both size and time estimation error dropped below 15% for half the class, with several student estimates within 3-5% of their actual values. For example, one size estimate of 507 LOC was off by only 11 LOC. One time estimate of 14.5 hours was off by only 25 minutes.

- Two students out of ten during the case study achieved what we consider to be the "Holy Grail" of PSP: 100% yield, i.e. programs that compiled and ran correctly the first time without any syntax or run-time errors.

The qualitative outcomes were equally positive. Most students expressed a very high degree of satisfaction with the course. The following comments are typical:

- "In September, I didn't know anything about software engineering. Now I know a great deal thanks to PSP. I now know the importance of why a process is used to finish a task. Software development is not easy and using a process helps in development."

- "I thought I was a good programmer, but after using PSP I realize that I was nothing back then. Now, I can proudly say that I have gotten much much better than ever before."

- "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 ... Teachers who push doing planning and design might actually know what they're talking about."

- "At the beginning, I just coded to finish the project or solve the problem. Now I take an in-depth look at the problem and think about it for a while before trying to develop a solution. By executing and learning this process I know way more about software engineering than when I started this course."

5.2. Data Quality Results

Despite these excellent educational outcomes, comparison of the original PSP dataset with the corrected dataset yielded 1539 errors. The following sections provide a breakdown of these defects according to their type, severity, age, the manner in which they were detected, whether they occurred during the analysis or collection stage, their "ripple effect", and the overall percentage error rate.

5.2.1. Error Types

We found that the errors naturally fell into one of seven general types. We present each type in descending order of frequency, and include the number of errors found of that type and the percentage of all errors represented by this type.

Calculation Error

(705 errors, 46%). This error type applied to data fields whose values were derived using any sort of calculation from addition to linear regression. If the calculation was not done correctly, an error was counted. This type was not used for values that were incorrect because fields used in the calculation contained bad numbers.

Blank Field

(275 errors, 18%). This error type was used when a data field required to contain a value, such as the *Start* field in a Time Recording Log entry, was left blank. This type was not used in fields where a value was optional, such as comment fields.

Transfer of Data Between Projects Incorrect

(212 errors, 14%) This error type was used for incorrect values in fields that involved data from a prior project. Typically these fields were "to date" fields that involved adding a to date value from a prior project with a similar value in the current project. Unfortunately, it was often impossible to determine if the error arose from bringing forward a bad number, or incorrectly adding two good numbers, or bringing forward the correct number and correctly adding it to the wrong number from the current form. However, in two important areas, time and size estimation, the forms were modified so that students were required to fill in the prior values to be used in the estimation calculations. In these cases we could determine when incorrect values originated in the transfer.

Entry Error

(142 errors, 9%). This error type applied when a student clearly did not understand the purpose of a field or used an incorrect method in selecting data. Examples include the use of a phase name in the *Fix Defect* field of the Defect Recording Log, or having the *Defects Injected, To Date* values in the Project Plan Summary originate from a different project than the *Program Size (LOC), To Date* values.

Transfer of Data Within Project Incorrect

(99 errors, 6%). This error type is similar to the error type involving incorrect transfer of data between projects, except that it applied to values transferred from one form to another within the current project. For example, filling in 172 for *Estimated New and Changed LOC* on the Size Estimating Template, but using 290 for *Total New and Changed, Plan* on the Project Plan Summary.

Impossible Values

(90 errors, 6%). This error type indicates that two values were mutually exclusive. Examples of this error type include overlapping time log entries, defect fix times for a phase adding up to more time than the time log entries for the phase, or phases occurring in the Defect Recording Log in a different order than those in the Time Recording Log.

Process Sequence Not Followed

(16 errors, 1%). This error type was used when the Time Recording Log showed a student moving back and forth between phases such as Compile and Test instead of sequentially moving through the phases appropriate for the process.

5.2.2. Error Severity

Some PSP data errors have relatively little "ripple effect" upon other data values, while others can have an enormous impact. To gain insight into the distribution of the ripple effect, we classified the errors into one of five "severity" levels. We present the levels in increasing order of ripple effect. As before, we include the total number of errors found for a given severity level and its percentage of the total.

Error Has No Impact on PSP Data

(104 errors, 7%). This level included errors such as missing header data, incorrect dates in the time recording log, and filling in fields for a more advanced process.

Results in a Single Bad Value, Single Form

(674 errors, 44%). This level was used if a significant field which affected no other fields, such as *LOC/Hour, Actual*, was blank or incorrect.

Results in Multiple Bad Values, Single Form

(197 errors, 13%). This level indicates when an incorrect or blank value was used in the calculation of values for one or more other fields on the same form, but when none of these other values were used beyond the current form. For example, in PSP1 on the Size Estimating Template, incorrectly calculating a prediction interval. This results in a bad prediction interval and a bad prediction range, but these values are not used anywhere else in the process.

Results in Multiple Bad Values, Multiple Forms, Single Project

(41 errors, 3%). This level indicates when an incorrect or blank value was used to determine the values for one or more other fields on one or more different forms in the same project, but when none of these other values were used beyond the current project. For example, in PSP1, on the Size Estimating Template, calculating an incorrect value for *Estimated Total New Reused (T)*. This results in an incorrect value for *Total New Reused, Plan* on the Project Plan Summary form, but this value is not referenced by future projects.

Results in Multiple Bad Values, Multiple Forms, Multiple Projects

(523 errors, 34%). This level was used if an incorrect or blank value affected future projects. For example, when *Defects Injected, Planning, Actual* on the Project Plan Summary does not match the number of defects entered for the planning phase in the Defect Recording Log.

5.2.3. Age of Errors

In any learning situation, a certain number of errors are to be expected. We hypothesized that perhaps the errors we discovered were simply a natural by-product of the learning process, and would "go away" as students gained experience with the various techniques in the PSP.

To evaluate this hypothesis, we calculated the "age" of errors—in other words, the number of projects since the introduction of the data field in which the error could be observed. If the errors were simply a by-product of the learning process, then we would expect a low average "age" for errors. In other words, people might make an error in a field initially, but then stop making the error after gaining more experience with the data field in question.

For example, the calculation of *Delta Time* for the Time Recording Log was introduced in the first project. If a student made an error in this field during the first project the error would have an age of zero. If a similar error was made during the second project the error would have an age of one. By the ninth project this type of error would have an age of eight.

We first analyzed the errors to determine the average error age in each project. Figure 2 shows the average age for all errors in each project.

Project #	PSP Process	# of Errors	Average Age
1	PSP0	51	0.00
2	PSP0.1	59	0.73
3	PSP0.1	63	1.76
4	PSP1	150	1.27
5	PSP1	165	2.27
6	PSP1	186	3.30
7	PSP1.1	160	3.26
8	PSP2	351	3.04
9	PSP2	354	3.84

Figure 2. Average error age by project—all errors.

Project #	PSP Process	# of Errors	Average Age
1	PSP0	0	NA
2	PSP0.1	43	1.00
3	PSP0.1	63	1.76
4	PSP1	70	2.71
5	PSP1	165	2.27
6	PSP1	186	3.30
7	PSP1.1	135	3.86
8	PSP2	214	4.99
9	PSP2	354	3.84

Figure 3. Average error age where age is not zero.

We then filtered out the 309 errors with an age of zero. This eliminated errors that could result from students being introduced to new fields and/or PSP operations for the first time. Figure 3 shows the resulting data.

When combining the 1539 errors from all projects, the average error age was 2.78 projects. After removing the 309 errors with an age of zero, the average error age rose to 3.48 projects.

Description	#
Time Estimation: historical data not transferred correctly	61
Size Estimation: historical data not transferred correctly	56
Time Log: delta time incorrect	48
Project Plan Summary: Total LOC, actual, not equal to B-D+A+R	45

Figure 4. Most frequently occurring persistent errors.

5.2.4. Error Detection Methods

In this study, there were three ways an error could be detected: by another student during technical review (40 errors), by the instructor during the grading/evaluation process (32 errors), or through the use of the PSP data entry tool (1467 errors). Thus, students were made aware of about 5% of the mistakes in their completed projects during the course of the class.

5.2.5. Analysis Stage Errors

Our two stage model of PSP data quality indicates that errors can be introduced during either collection or analysis. Most of the errors that we detected occurred during PSP analysis activities, with 700 errors occurring in the Plan phase and 561 errors in the Postmortem phase. Some of the errors occurring in other phases, such as errors in *Delta Time* calculations, were also analysis errors.

The Most Severe Errors.

34% of errors found were of the most serious type—persistent errors. These were the errors resulting in multiple bad values on multiple forms for multiple projects. A defect of this type not only causes incorrect values in the current project, but may still be causing flawed results ten projects later, even if all subsequent calculations are done correctly. Figure 4 shows the four most common errors of this type.

There were two main ways that the error in transferring time estimation data appeared to occur: incorrectly transferring the value from the correct field, or accidentally transferring the correct value from an incorrect field. For example, instead of transferring *Total New and Changed (N)* (Plan or Actual), students often transferred *Total LOC (T)*. This could easily

Project #	Errors	Time Log Entries	% in Error
1	7	84	8.33
2	2	88	2.27
3	8	92	8.70
4	8	108	7.41
5	2	102	1.96
6	9	121	7.44
7	2	77	2.60
8	5	122	4.10
9	5	105	4.76

Figure 5. Delta time errors by project.

occur because the Project Plan Summary form has over 90 fields even at the level of PSP1, and these two values are vertically adjacent on the form. It is particularly easy to make this mistake with the Actual values because the fields are separated by one column from the labels. Additionally, it appeared that students made spreadsheets to avoid thumbing through the entire stack of completed projects every time a time or size estimation was needed for a new project. We infer this because the same incorrect value for a particular project would be transferred over and over again for time and/or size estimation in new projects.

Similar factors surround the error in transferring data for size estimation. These transfer errors were not insignificant. Over the 56 errors resulting from incorrect transfer of data used for size estimation, the sum of the errors was 7753 LOC (lines of code), with an average error of 138.4 LOC. The sum of the LOC as they should have been transferred was 10,255, with an average of 183 LOC per field. Thus, the average incorrectly transferred number was in error by an amount equaling 75.6% of the number that should have been transferred.

The error in calculating *Delta Time* in the Time Recording Log was notable in several respects. First, the errors were not insignificant. The average mistake was 37.8 minutes, which was an average of 39.9 percent of the correct value. Second, of 48 occurrences, 16 were in error by one hour and 4 were in error by two hours, indicating small errors in simple arithmetic. Third, the distribution of this error across projects is as shown in Table 5.

Despite nine projects worth of experience, this error never “went away”. However it did appear to occur less frequently after Project 6. Interestingly, the assignment for this project was a Time Recording Log applet, which at least some students seem to have used for subsequent projects.

5.2.6. Collection Stage Errors

As noted previously, analysis stage errors are relatively easy to determine and correct. However, the accuracy of recorded process measures from the collection stage was much more difficult to examine because the time of collection had already passed and, unlike the analysis operations, was impossible to reproduce. However, we found both direct and indirect evidence for collection errors during the case study.

Direct Collection Error Evidence

Direct evidence of collection problems appeared in the 90 errors of type of "Impossible Values". We classified these errors into three major subtypes.

Internal Time Log Conflicts.

There were five time logs with overlapping entries, indicating some sort of problem with accurately collecting time-related data.

Internal Defect Log Conflicts.

51 errors showed problems with correctly collecting defect data. 48 of these errors were Defect Recording Log entries showing defects that were injected during the Compile and Test phases, but these same defects were not noted as being the result of correcting other defects found during Compile or Test.

Discrepancies Between Time and Defect Logs.

In 22 cases, Defect Recording Log entries were entered with dates that did not match any Time Recording Log entries for the given date. For example, a defect would be recorded as removed during the Code phase on a Wednesday, but the time log would show that all coding had been completed by Monday and that the project was in the Test phase on Wednesday. For 10 projects, the total *Fix Time* for defects removed during a particular phase added up to more time than was recorded for that phase in the Time Recording Log. Finally, in two cases, the Defect Recording Log showed a different phase order than the Time Recording Log.

Indirect Collection Error Evidence

Besides the recorded errors, there were other indicators that collection problems had occurred. Some Time Recording Logs showed a suspicious number of even-hour (e.g. 6:00 to 7:00, 10:00 to 12:00) entries, even though students were required to record times at

the minute level. Others showed long stretches of consecutive entries with no breaks or interruptions. Often, the total *Fix Time* for the defects in a phase was far less than the time spent in the phase. For example, the Time Recording Log might show three hours spent in the Test phase, but the Defect Recording Log would show two defects that took eight minutes to fix. Obviously, it is not impossible that this would occur, but it is much more likely that not all defects found in test were recorded.

In a similar vein, some projects had suspiciously few defects overall, such as seven defects for a project with 284 new lines of code and almost 11 hours of development time, (including 40 minutes in compile for two defects requiring 6 minutes of fix time). Our analysis of the PSP data for that same project yielded 27 errors.

Finally, the instructor has anecdotally observed the following trend in every PSP course he has taught so far: the students turning in the highest quality projects also tend to record far higher numbers of defects than the students who turn in average or lower quality projects. If this trend is real, then we can provide two possible explanations. It may be the case that the students turning in lower quality projects tend to make far fewer errors than those turning in the higher quality projects, although this seems *extremely* unlikely. What appears more likely is that the students turning in the highest quality projects also exhibit the lowest level of collection error, which indicates that substantial but non-enumerable collection error exists in the PSP data we examined.

5.2.7. Comparison of Original and Corrected Data

When we compared the original and corrected data, we found significant differences ($p<.05$) for the Cost-Performance Index (planned time-to-date/actual time-to-date) and Yield (percentage of defects injected before first compile that were also removed before first compile). We used the Wilcoxon Signed Rank Test (Ferguson and Takane, 1989), a non-parametric test of significance which does not make any assumptions regarding the underlying distribution of the data. Figure 6 and Figure 7 illustrate the differences between these two measures graphically.

A CPI value of 1 indicates that planned effort equals actual effort. CPI values greater than 1 indicate overestimation of resource requirements, while CPI values less than 1 indicate underestimation of resource requirements. In half of the subjects, correction of the CPI value reversed its interpretation (from underplanning to overplanning, or vice-versa). In the remaining cases, several corrected CPI values differed dramatically from original values. For example Subject A's original CPI was 0.32, indicating dramatic underplanning, while the corrected CPI was 0.99, indicating an average planned resource requirements virtually equal the average actual resource requirements.

Correction of yield values tended to move their values downward, sometimes dramatically. In half of the subjects, the corrected yield was less than half of the original yield values, indicating that subjects were removing a far fewer proportion of defects from their programs prior to compiling than indicated by the Yield measurement.

These particular results confirm our hypothesis. In the case of CPI, use of the uncorrected data would lead half of the subjects to make exactly the wrong process improvement.

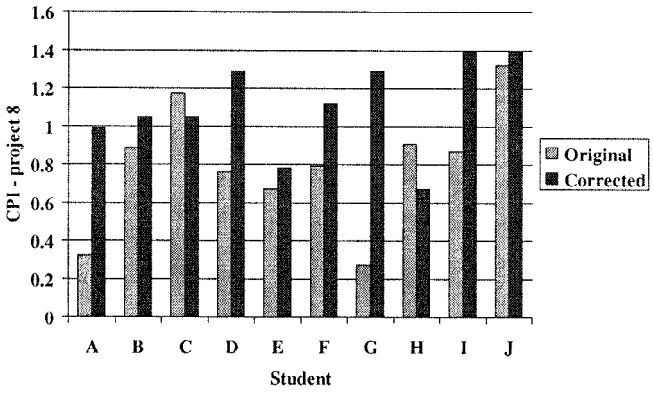


Figure 6. Effect of correction on CPI.

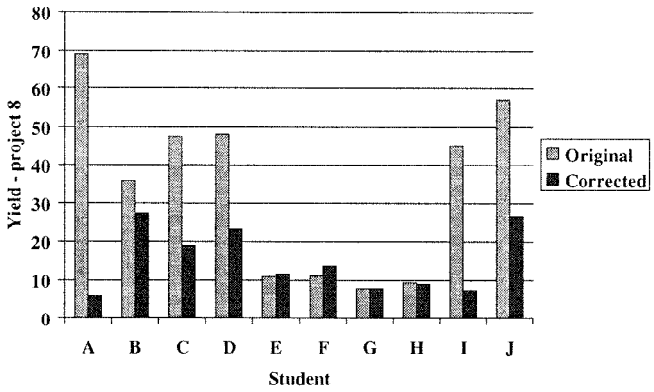


Figure 7. Effect of correction on yield.

In the case of yield, use of the uncorrected data would lead subjects to not take process improvement actions indicated when yield is low.

5.2.8. Overall Percentage Error Rate

Such a large number of data quality errors calls into question the quality of instruction. Perhaps these results are a simple artifact of poor quality control on the part of the teacher? Unfortunately, the very large number of data values to check in the manual PSP suggests otherwise.

For example, a time recording log contains six fields (plus a comment field, but for our purposes, this field is extraneous): *Date, Start, Stop, Interrupt time, Delta Time*, and *Phase*.

Process	Approx. Fields	Projects	Total Values
PSP0	200	10	2000
PSP0.1	220	20	4400
PSP1.0	329	20	6580
PSP1.1	437	20	8740
PSP2.0	528	19	10,032
	Total	**89**	**31,752**

Figure 8. Data values present in PSP.

Students typically entered about 10 time log entries for an assignment. This results in 60 data values to check for one student on one assignment, and 600 data values to check for a class of 10 students. This is for one form and one assignment. Following this approach, one can arrive at an estimate of almost 32,000 data values to be checked by hand for this single case study, as illustrated in Figure 8. The 1539 data errors uncovered during this study represents only 4.8% of the total possible, which means that the instructor obtained over 95% correctness (at least with respect to analysis-stage data quality).

6. Discussion

This paper reports on the results of analysis of the data from a single PSP class with only 10 students. As with any case study, care must be taken in interpreting these results. We do not know whether this data is representative of PSP courses in general, and if the way we teach the PSP is representative of the way the PSP is taught by others. Data quality problems might be less prevalent in other PSP courses; on the other hand, they might just as easily be more prevalent.

While we do not claim that these results are representative of all PSP courses, neither do we believe that they are an artifact of some peculiarity and/or failing of our environment. First, this case study was performed after the instructor had taught the PSP for one semester in a graduate level course, and instituted it within his research group, and adopted it himself for his own software development activities. By the time of this study, we were quite experienced as both teachers and users of the PSP. Second, as already noted, we were concerned with data quality problems from the beginning, and instituted curriculum modifications specifically intended to raise data quality. The overall error rate of less than 5%, while quite small, was still not sufficiently small to prevent significant differences between original and corrected data sets. Third, our results cannot be due to our lack of enthusiasm for the PSP: both of us consider it to be one of the most powerful software engineering practices we have adopted in our careers. The second author, for example, has used her automated PSP tool to gather data on over 120 of her industrial projects over the past two years. Fourth, our

results cannot be due to lack of enthusiasm for the PSP by our students, as the post-course comments reveal, most of the students indicated that they found the class to be very useful and interesting.

6.1. Recommendations for Research and Practice

Based upon the results of this study, we have the following recommendations for future research and practice of the PSP:

6.1.1. Replication

We believe this study provides strong evidence for the need for more research on collection and analysis data quality in the PSP. Current studies of the PSP appear to take the accuracy of PSP data for granted, or else simply assume that tool support can eliminate all sources of data quality problems. This study is the first to methodically examine the assumptions underlying data quality in the PSP and subject them to empirical investigation. Our results indicate that the PSP community may be overly optimistic about the quality of PSP data, particularly when produced using the traditional, manual approaches that lack integrated, PSP-specific tool support. Even when such support is provided, the possibility of measurement dysfunction introduces substantial threats to the accuracy of the data in the collection phase as discussed below in Section 6.1.5. Better understanding of the true extent of PSP data quality problems requires replication of this study, or at least further PSP research that includes PSP data quality verification as an explicit design component. To support this endeavor, researchers are invited to peruse a website containing curriculum materials from this course at http://www.ics.hawaii.edu/~johnson/613s98/.

6.1.2. Software Engineering Education

We continue to believe that the PSP has substantial educational value in software engineering, despite the issues we have raised with data quality. Students learn valuable, concrete skills concerning defect management and planning in the PSP curriculum. Additionally, the PSP provides students with a framework for empirically evaluating the usefulness of any other process improvement techniques or programming methods they come across in the future.

6.1.3. PSP Tool Support

We believe that integrated tool support for the PSP is required, not merely helpful, to obtain high analysis-stage PSP data quality. We also believe that integrated tool support will make adoption of the PSP substantially easier, since the most common complaint made by students using the manual PSP in our classes is the time and effort required to fill out the forms. Currently, we have designed and implemented a Java-based toolset for integrated empirical

software process improvement that automates many of the analysis stage computations in the PSP, and which extends the PSP paradigm with support for group review and patterns (Moore, 1998). We are currently using this toolset, called Leap, in a software engineering course and will deploy it in an extensively redesigned PSP-like curriculum in Fall 1999.

6.1.4. PSP Research Design

We believe that the results of this case study have a number of implications for current and future research on the PSP.

First, until questions raised by this study with respect to PSP data quality are resolved, PSP data should not be used to evaluate the PSP method itself. In other words, we believe that it is not yet appropriate to assume that changes in PSP measures during (or after) a training course accurately reflect changes in the underlying developer behavior. A statement such as "The improvement in average defect levels for engineers who complete the course is 58.0%", if based upon PSP data alone, might only reflect a decreasing trend in defect recording, not a decreased trend in the defects present in the work product.

Second, our research on the PSP has demonstrated that high quality pedagogical design is not equivalent to high quality experimental design. In other words, some of the features of the PSP with respect to pedagogy are bugs with respect to experimental design.

One problem in the PSP with respect to experimental design concerns uncontrolled instrumentation. The PSP programming exercises incrementally build a set of tools for use in gathering and managing PSP data. This is elegant pedagogically, since it enables an instructor to use the PSP and have the students build partial tool support for it as they go along. Unfortunately, this is disastrous from an experimental design viewpoint, since it means that crucial primary data measures (size) and derived measures (size and time estimates) are calculated from a set of student programs with no experimental control over their quality and accuracy. We know from bitter experience that writing a high quality size counting and differencing tool for Java that handles all aspects of the language and produces both a meaningful measure of size and differences in size between two versions of a program is a nontrivial programming project. It requires extensive design, implementation, and field use far beyond the 10 days available for this program in the PSP curriculum. For the PSP curriculum to be useful experimentally, there must be control over and verification of the instrumentation.

Another problem in the PSP with respect to experimental design concerns the lack of control over curriculum modifications. For example, the SEI study notes that "there are many cases where instructors tailored the training course (including selection of assignments, data collection requirements, and sequence of introduction for process changes.)" Our course also deviates from the standard curriculum.

Yet another problem in the manual PSP with respect to experimental design concerns systematic bias in the data. For example, the PSP curriculum requires, in an academic setting, a full semester course. In academic settings, the workload on students tends to be light during the beginning of the semester, become heavier after midterms, and reach a peak near the end of the semester. For PSP measures to be accurate, students must maintain a consistent level of process data collection throughout the course of the semester. From our

personal experience, we have observed that a portion of the students in our PSP classes appear to begin to "cut corners" in their recording of defects and time near the end of the semester, presumably due to external pressures on their time and energies. This "end of semester crunch" can introduce a systematic bias into PSP data, leading to, for example, artificial decrease in defect density values near the end of the course.

Another example of systematic bias can occur from what we term the "process overhead ceiling effect". Many students complain that the amount of effort collecting and analyzing PSP process data can equal, exceed, or interfere with the time and focus required to actually develop the programs. Early in the course, process overhead consists almost purely of time and defect data collection, so students devote a great deal of time and energy to that task. By the end of the semester, the total process overhead of the PSP has risen dramatically, since estimation, time and schedule planning, and so forth have all been added. If at least some portion of the students decide to limit the amount of time spent on process collection and analysis, the most likely place to cut corners is, once again, in defect recording, which would once again produce an artificial decrease in defect density values near the end of the course.

A final example of systematic bias occurs from the format of the manual PSP forms themselves. As we note in our results, the case study students frequently transferred a "Total" LOC value from one form to another instead of the "New and Changed" LOC value. Since the Total value is always greater than "New and Changed", a systematic bias toward inflated system sizes is present. We found other situations in which the design of the forms lead to consistent user errors.

From an experimental design standpoint, uncontrolled instrumentation and systematic bias are threats to the internal and external validity of any study which both uses the manual PSP and which draws conclusions about underlying programmer behavior based purely upon the PSP data. One example of research suffering from these threats is the Software Engineering Institute technical report by Hayes and Over (1997). The report refers to collection of "paper forms", indicating the manual PSP. There is no mention of any control over the quality and accuracy of PSP instrumentation, such as the size counter. There is no mention of any rigorous validation of the PSP data. Instead, the researchers simply claim that "the quality and accuracy of the data used in any given class tend to be exceptional." Unfortunately, our case study shows that even an accuracy of over 95% in the PSP dataset is insufficient to obtain data accurately reflecting underlying programmer behavior. Furthermore, our original dataset is quite consistent in its outcome with the aggregate outcome reported by the SEI. The research design presented in their report cannot detect the data quality problems in our original dataset, and so presumably cannot detect data quality problems present in any of the datasets actually used in the study. Finally, although the researchers subjected the PSP data to extensive statistical analysis, these analyses all assume the absence of systematic bias in the dataset, an assumption which we believe to be incorrect in the manual PSP.

We are happy to note that not all PSP evaluations are based upon PSP data alone. For example, in one industrial case study, evidence for the utility of the PSP method was based upon reductions in acceptance test defect density for products subsequent to the introduction of PSP practices (Ferguson et al., 1997). Although alternative explanations for this trend can be hypothesized (such as the PSP-based projects were more simple than those before and

thus acceptance test defect density would have decreased anyway), at least the evaluation measure is independent of the PSP data and not subject to PSP data quality problems.

6.1.5. Collection Data Quality and Measurement Dysfunction

Unfortunately, integrated tool support is not a "magic bullet" that will solve all PSP data quality problems. As our simple model of PSP data quality shows, no matter how perfectly we are able to automate the analysis stage, overall PSP data quality will still depend largely upon the data quality from the collection stage.

Our case study was able to detect substantial numbers of analysis errors which could be eliminated through appropriate automation. Our case study was also able to detect the potential presence of substantial collection errors, but the solution to this issue is much more complex. It is currently beyond the state of the art to accurately and completely automate the collection of all primary process measures (time, size, defects) for a programmer. For the foreseeable future, we must rely on users of the PSP to accurately and consistently record primary data values.

In our research on the collection data quality problem, we have gained insight from research on "measurement dysfunction" (Austin, 1996). According to Austin, whenever you measure an attribute of an organization with the goal of improving the organization's performance, you run the risk of worsening the organization's performance as a direct result of the measurement. This is because there are at least two uses to which a given measurement can be applied: for information and for motivation.

Informational measurement "tells about an organizational process . . . It is used to learn from and to plan." In the PSP, all measures are intended to be informational.

Motivational measurement, on the other hand, "is used to quantify the value of compensation for compliance with objectively verifiable standards of work." In other words, motivational measurement is used to evaluate the performance of individuals. In the PSP, no measures are meant to be motivational.

Although this seems straightforward, a principal claim of Austin's research is that any individual measure is "value-free" with respect to its application: it can be used for informational purposes, motivational purposes, or both. Importantly, it is impossible for an organization to guarantee that a measure, once collected, will never be used for motivational purposes. Thus, individuals in an organization may tend to operate under the assumption that any measures of individual performance can be used for motivational purposes, regardless of the stated intention of the organization with respect to that measure at the time it is taken.

We find the measurement dysfunction perspective quite revealing with respect to the PSP, because in any PSP academic or industrial training situation, the "organization" collects the PSP measures from the individual. Even though competant PSP instructors always inform the students that they will not be evaluated on the actual values of their PSP data they collect, measurement dysfunction theory indicates that individuals may still act under the assumption that they might at some point become accountable for the values they submit. As PSP data provides very revealing and potentially dangerous information about a programmer's practice, the appropriate PSP data to provide the organization for motivational measurement may be quite different from the appropriate data for personal, informational measurement.

We conjecture that collection stage data quality requires, at a minimum, a combination of low collection overhead along with environmental features that minimize the potential for measurement dysfunction. Overhead can be reduced through tool support that makes manual recording of time, defect, and size data fast and accurate. Minimizing measurement dysfunction requires, in essence, the property of privacy for PSP data—in other words, that the organization does not and cannot have access to an individual's PSP data.

Measurement dysfunction, unfortunately, introduces yet another obstacle to the use of PSP data for experimental purposes. In order to teach a PSP course effectively, the instructor must inspect the PSP data submitted by students. However, this essential educational feature violates the privacy of an individual's PSP dataset, an essential feature to minimize measurement dysfunction. The problem of measurement dysfunction, on top of the problems cited earlier, lead us to question if collecting PSP data from an educational setting is a fundamentally unsound approach to assessing underlying programmer behavior. If this is true, we must redesign our current paradigms for research using the PSP.

7. Acknowledgments

We gratefully acknowledge all of the students in all of the PSP classes at the University of Hawaii. Our colleagues in the Collaborative Software Development Laboratory during the time of this research (Cam Moore, Robert Brewer, Jennifer Geis, Joe Dane, and Russ Tokuyama) provided ongoing support. We would like to thank Watts Humphrey, James Over, and Will Hayes of the Software Engineering Institute, and the anonymous reviewers, whose comments sharpened the presentation of this research. This research was sponsored in part by grants CCR-9403475 and CCR-9804010 from the National Science Foundation.

References

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

Ceberio-Verghese, A. 1996. Personal Software Process: A user's perspective. In Nancy R. Mead, editor, *Ninth Conference on Software Engineering Education*, 10662 Los Vaqueros Circle, P. O. Box 3014, Los Alamitos, CA 90720-1264. IEEE Computer Society Press.

Dellien, O. 1997. The Personal Software Process in industry. Master's thesis, Lund Institute of Technology (Sweden), Department of Communication Systems.

El Emam, K., Shostak, B., and Madhavji, N. 1996. Implementing concepts from the Personal Software Process in an industrial setting. In *Proceedings of the Fourth International Conference on the Software Process*, Brighton, England.

Ferguson, G. A., and Takane, Y. 1989. *Statistical Analysis In Psychology And Education*. McGraw-Hill Book Company, 6th edition.

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

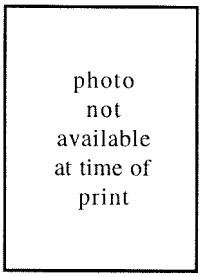
Gilb, T., and Graham, D. 1993. *Software Inspection*. Addison-Wesley.

Hayes, W., and Over, J. W. 1997. The Personal Software Process (PSP): An empirical study of the impact of PSP on individual engineers. Technical Report CMU/SEI-97-TR-001, Software Engineering Institute, Pittsburgh, PA.

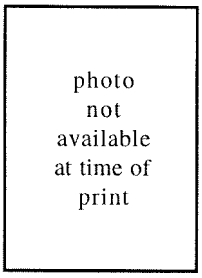
Henry, J. 1997. Personal Software Process studio. http://www-cs.etsu.edu/softeng/psp/.

Humphrey, W. 1994a. The personal process in software engineering. In *Proceedings of the Third International Conference on the Software Process*, 69–77.

Humphrey, W. 1994b. The Personal Software Process. *Software Process Newsletter #1*, pages 1–3.
Humphrey, W. 1995. Introducing the Personal Software Process. *Annals of Software Engineering* 1: 311–325.
Humphrey, W. 1995. The power of personal data. *Software Process Improvement and Practice Journal* 1: 69–81.
Humphrey, W. S. 1995. *A Discipline for Software Engineering*. New York: Addison-Wesley.
Humphrey, W. S. 1996. Using a defined and measured Personal Software Process. *IEEE Software* 13(3): 77–88.
Humphrey, W. S. 1997. *Introduction to the Personal Software Process*. New York: Addison-Wesley.
Moore, C. 1998. Project LEAP toolset. http://csdl.ics.hawaii.edu/Tools/LEAP/LEAP.html.
Paulk, M., Weber, C., Curtis, B., and Chrissis, M. B. 1995. *The Capability Maturity Model: Guidelines for Improving the Software Process*. New York: Addison-Wesley.
Sherdil, K., and Madhavji, N. H. 1996. Human-oriented improvement in the software process. In *Proceedings of the 5th European Workshop on Software Process Technology*.
Wohlin, C. 1998. The Personal Software Process as a context for empirical studies. *Software Process Newsletter #12*, pages 7–12.



Philip M. Johnson received the BS degrees in biology and computer science from the University of Michigan, and the MS and PhD degrees in computer science from the University of Massachusetts. Dr. 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. His research interests include computer supported cooperative work, software quality assurance, and software process improvement. He is a member of the IEEE Computer Society and the ACM.



Anne M. Disney received a BS degree in Computer and Information Science from the University of Maryland and an MS degree in Information and Computer Sciences from the University of Hawaii. She is currently working as a software engineer for Infoworld Management Systems.