The Personal Software Process℠ (PSP℠): An Empirical Study of the Impact of PSP on Individual Engineers

Will Hayes
James W. Over

December 1997

Technical Report
CMU/SEI-97-TR-001
ESC-TR-97-001

Technical Report
CMU/SEI-97-TR-001
ESC-TR-97-001
December 1997

The Personal Software Process[SM] (PSP[SM]): An Empirical Study of the Impact of PSP on Individual Engineers

Will Hayes

James W. Over

Software Engineering Measurement and Analysis Initiative
Personal Software Process Initiative

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Jay Alonis, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Asset Source for Software Engineering Technology (ASSET): 1350 Earl L. Core Road; PO Box 3305; Morgantown, West Virginia 26505 / Phone:—(304) 284-9000 / FAX—(304) 284-9001 World Wide Web: http://www.asset.com / e-mail: sei@asset.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone—(703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218 / Phone—(703) 767-8274 or toll-free in the U.S.—1-800 225-3842.

Table of Contents

List of Figures

List of Tables

Acknowledgments

The authors of this report wish to extend their gratitude to the 298 software engineers who worked to collect the data analyzed here, as well as the instructors and organizations that provided this data to us.

The reviewers of our initial draft (many of whom are in the group above) provided invaluable feedback leading to improvements in this report. For this feedback, we thank

Peter Abowd	Bob Musson
Dan Burton	Mark Paulk
Khaled El Emam	Bill Peterson
Wolf Goethert	Marsha Pomeroy-Huff
Dennis Goldenson	Bob Powels
Tom Hilburn	Dan Roy
Andy Huber	Jeff Schwalb
Watts Humphrey	Barry Shostak
Mike Konrad	David Silverberg
Jim McCurley	Dave Zubrow

Michael Zuccher provided database support, and a strong sense of team spirit in the creation of a single database from over 300 spreadsheet files. We are grateful for his hard work.

We wish to thank Bob Lang, Marsha Pomeroy-Huff, Bill McSteeen, and Pennie Walters for their assistance in editing and revising this report.

Jim McCurley and Dave Zubrow provided consultation and feedback regarding the many statistical issues confronted during the analyses. Their contributions in suggesting and validating the choice of statistical methods are much appreciated.

Finally, for his continued pioneering work in the field of software engineering, we wish to recognize Watts Humphrey. His vision and tireless effort made this work possible.

The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers

Abstract: This report documents the results of a study that is important to everyone who manages or develops software. The study examines the impact of the Personal Software Process℠ (PSP℠) on the performance of 298 software engineers.[1] The report describes the effect of PSP on key performance dimensions of these engineers, including their ability to estimate and plan their work, the quality of the software they produced, the quality of their work process, and their productivity. The report also discusses how improvements in personal capability also improve organizational performance in several areas: cost and schedule management, delivered product quality, and product cycle time.

1 Executive Summary

The PSP is a defined and measured software process designed to be used by an individual software engineer. The PSP was developed by Watts Humphrey and is described in his book *A Discipline for Software Engineering* [Humphrey 95]. Its intended use is to guide the planning and development of software modules or small programs, but it is adaptable to other personal tasks.

Like the SEI Capability Maturity Model℠ for Software,[2] the PSP is based on process improvement principles. While the CMM® is focused on improving organizational capability,[3] the focus of the PSP is the individual engineer. To foster improvement at the personal level, PSP extends process management and control to the software engineer. With PSP, engineers develop software using a disciplined, structured approach.

They follow a defined process, plan, measure, and track their work, manage product quality, and apply quantitative feedback to improve their personal work processes, leading to

- better estimating
- better planning and tracking

1. PSP and Personal Software Process are service marks of Carnegie Mellon University.

2. Capability Maturity Model is a service mark of Carnegie Mellon University.

3. CMM is registered in the U.S. Patent and Trademark Office.

- protection against overcommitment
- a personal commitment to quality
- the engineers' involvement in continuous process improvement

Thus, both the individual and the organization's capability are improved.

1.1 Study Results

In this study we examined five personal process improvement dimensions of the PSP: size and effort estimation accuracy, product quality, process quality, and personal productivity. We found that the PSP improved performance in the first four of these dimensions without any loss in the fifth area, productivity.

- Effort estimates improved by a factor of 1.75 (median improvement).
- Size estimates improved by a factor of 2.5 (median improvement).
- The tendency to underestimate size and effort was reduced. The number of overestimates and underestimates were more evenly balanced.
- Product quality, defects found in the product at unit test, improved 2.5 times (median improvement).
- Process quality, the percentage of defects found before compile, increased by 50% (median improvement).
- Personal productivity, lines of code produced per hour, did not change significantly. However, the improvement in product quality resulting from the PSP is expected to improve productivity and cycle time as measured at the project level (i.e., when integration and system test phase effort are included in productivity and cycle time).

These study results have significant implications for any organization that develops software. They indicate that the PSP can help software engineers achieve statistically significant improvements in four areas that are of critical importance from a business perspective: cost and schedule, product quality, cycle time, and organizational process improvement. Early results from three industry case studies support this conclusion [Ferguson 97].

1.1.1 Cost and Schedule Management

A critical business need for all organizations that develop software is better cost and schedule management. Cost and schedule problems often begin when projects make commitments that are based on inadequate estimates of size and development resources. PSP addresses this problem by showing engineers how to make better size and resource estimates using statistical techniques and historical data. Estimates with the PSP were more accurate, and equally important, an initial bias towards underestimating shifted to a more balanced mix of overestimates and underestimates. A more balanced estimation error means that the errors tend to cancel, rather than compound, when multiple estimates are combined.

1.1.2 Quality Management

A second critical business need is for improved software quality. Poor quality management now limits our ability to field many critical systems, increases software development costs, and makes development schedules even harder to predict. Many of these quality problems stem from the practice of relying on testing to manage software quality. But finding and fixing defects in test is costly, ineffective, and unpredictable.

The most efficient and effective way to manage software quality is through a comprehensive program focused on removing defects early, at the source. The PSP helps engineers to find and remove defects where they are injected, before compile, inspection, unit test, integration test, or system test. With fewer defects to find and remove in integration and system test, test costs are reduced sharply, schedules are more predictable, fewer defects are released to the field, maintenance and repair costs are reduced, and customer satisfaction is increased.

1.1.3 Cycle Time

A third critical business need is for reduced product cycle time. Cycle time can be reduced through better planning and the elimination of rework through improvements in product quality. Accurate plans allow for tighter scheduling and greater concurrency among planned activities. Better quality through early defect removal reduces waste and further increases planning accuracy by reducing a source of variation, the discovery and repair of defects in integration and system test.

With PSP, engineers learn how to gather the process data needed to minimize cycle time. The data helps them to build accurate plans, eliminate rework, and reduce integration and system test by as much as four to five times [Ferguson 97].

1.1.4 Organizational Process Improvement

A fourth critical business need is process improvement. It is well understood that process improvement can increase competitive advantage, but it has been difficult to involve software engineers. They often view process improvement and product quality as a management or staff activity, not as their personal responsibility. With PSP, engineers gain personal experience with process improvement. They become process owners, directly involved in the measurement, management, and improvement of the software process.

1.2 PSP Introduction

Successful introduction of the PSP requires sponsorship and participation by all management levels. An effective strategy is to first involve key executives and managers, then to begin training engineers in the PSP, implementing on a project-by-project basis.

The most significant cost in introducing the PSP is training the engineers. Engineers will typically spend from 125 to 150 hours to complete PSP training over a period of one to three months. This investment can be quickly recovered through reductions in test and rework costs, increased productivity,[4] and efficiencies resulting from more accurate plans. A project team of six engineers developing a product of at least 15 KLOC will typically save enough time in integration and system test on their first project to cover the cost of PSP training.

1.3 About the Study

The objectives of this study were to test key assertions about the benefits of the PSP and to consider whether the observed results can be generalized beyond the study participants. Because the PSP was developed to improve individual performance, the study examined changes in individual performance as new practices were introduced. Though changes in the group average are reported, the focus of the study has been on the average change for individuals in the group, rather than on the change in the group average. Using this model, each person provided his/her own baseline to the analysis, so variation among individuals does not 'contaminate' the statistical results.

The report includes detailed presentations of the statistical analyses conducted on size and effort estimation accuracy, process yield, defect density, and productivity. The report also includes other observations uncovered during the statistical analysis and study conclusions. Preceding the analysis sections is an overview of the PSP, the PSP process, and the PSP course. A detailed description of the PSP basic measures,[5] planning and measurement forms, and the development and data collection processes are also included to provide additional context for understanding the results.

As the analyses show, the PSP improved personal performance in four of five dimensions studied. We know that similar results can be achieved in practice [Ferguson 97] and encourage you to consider the potential impact on your organization's performance.

4. Personal productivity was unchanged, but product cycle time and project productivity should improve due to reductions in integration and system test cost resulting from improved quality at the PSP level.

5. These are time (effort), defects, and size.

2 Introduction and Background

All businesses are becoming software businesses. That is, more and more businesses now develop and incorporate software in the products they produce, or develop software to support the design, manufacture, or delivery of the products and services they provide. Many are finding that as the software component of their business grows, schedule delays, cost overruns, and quality problems caused by software are becoming their number one business problem. Suddenly, software development has become the high-risk element in their business plans. Moreover, despite their best management efforts, they find their risk of failure increasing with each increase in the size or complexity of the software they produce.

Software businesses have at least one thing in common. The business is dependent on people. Software products are made of hundreds to millions of individual computer instructions, each one handcrafted by a software engineer. And so, the technical practices and experience of their engineers largely determine the outcome of the development process, as has been widely recognized [Boehm 81].

Software businesses also share a common set of needs: better cost and schedule management, improved software quality, and reduced software development cycle time. PSP directly addresses these needs by improving the technical practices and individual abilities of software engineers, and by providing a quantitative basis for managing the development process. By improving individual performance, PSP can improve the performance of the organization, as individual improvements in aggregate are reflected at the organizational level.

PSP can improve the business of software development in several ways:

- Data from the PSP improve planning and tracking of software projects.
- Early defect removal results in higher quality products, as well as reductions in test costs and cycle time.
- PSP provides a classroom setting for learning and practicing process improvement. Short feedback cycles and personal data make it easier to gain understanding through experience.
- PSP helps engineers and their managers learn to practice quantitative process management. They learn to use defined processes and collect data to manage, control, and improve the work.
- Finally, PSP exposes engineers to 12 of the CMM Key Process Areas (KPAs). They are better prepared to participate in CMM-based improvement.

As the data illustrate, the PSP significantly improves personal performance during the PSP training course. We believe that widespread adoption of the PSP will produce the same results, and that similar benefits will be observed at the project and organizational levels.

2.1 The PSP Course

The PSP provides a framework for teaching engineers about the software process, and a starting point from which they can evolve their own personal processes. The PSP is based on the same industrial practices that are found in the SEI CMM, but scaled-down for individual use. It is a defined and measured, individualized process for consistently and efficiently developing high quality software modules or small programs. It has been adapted to many other kinds of software engineering tasks, such as developing software requirements, software specifications, and test cases. The PSP can also be scaled up to support small projects by integrating individual personal processes with a project process that was based on the PSP architecture. In general, once learned, the principles and concepts in the PSP can be applied to any structured, repetitive task.

The SEI is working to transition the PSP into software engineering education programs in both industrial and academic settings. We began offering PSP training in October 1994. Our initial offering was a train-the-trainer course. Since then, more than fifty PSP instructors have been trained to teach the PSP and five companies have been licensed to provide PSP training to software engineers in industry.[1] We have also provided on-site training to more than 200 engineers and managers.

The SEI is also working with academia to support the use of PSP in computer science and software engineering education at the graduate and undergraduate levels. In June of 1997, the SEI and Embry-Riddle Aeronautical University held the first PSP Faculty Workshop. The purpose of the workshop was to introduce university faculty to the PSP and the standard PSP course. The SEI plans to continue to support the academic community by holding similar workshops in the future.

The standard PSP course prepares an engineer to apply the PSP in practice. The course follows a staged learning strategy described in the textbook *A Discipline for Software Engineering* [Humphrey 95]. The text was designed for use in graduate and senior-level undergraduate courses. Because the textbook is self-contained, experienced engineers could use the textbook to help them learn the PSP on their own, but most engineers need the structure and support of a formal training course to complete the training.

The PSP course incorporates what has been called a “self-convincing” learning strategy that uses data from the engineer’s own performance to improve learning and motivate use. The course introduces the PSP practices in steps corresponding to seven PSP process levels. Each level builds on the capabilities developed and historical data gathered in the previous level. Engineers learn to use the PSP by writing ten programs, one or two at each of the seven levels, and preparing five written reports. Engineers may use any design method or programming language in which they are fluent. The programs are typically around one

1. See “SEI Transition Partners for Personal Software Process (PSP) Training” on the SEI Web site: http://www.sei.cmu.edu/participation/trans.part.psp.html

hundred lines of code (LOC) and require a few hours on average to complete. While writing the programs, engineers gather process data that are summarized and analyzed during a postmortem phase. With such a short feedback loop, engineers can quickly see the effect of PSP on their own performance. They convince themselves that the PSP can help them to improve their performance; therefore, they are motivated to begin using the PSP after the course.

2.1.1 The PSP Process Levels

The seven process levels used to introduce the PSP are shown in Figure 2-1. Each level builds on the prior level by adding a few process steps to it. This minimizes the impact of process change on the engineer, who needs only to adapt the new techniques into an existing baseline of practices.

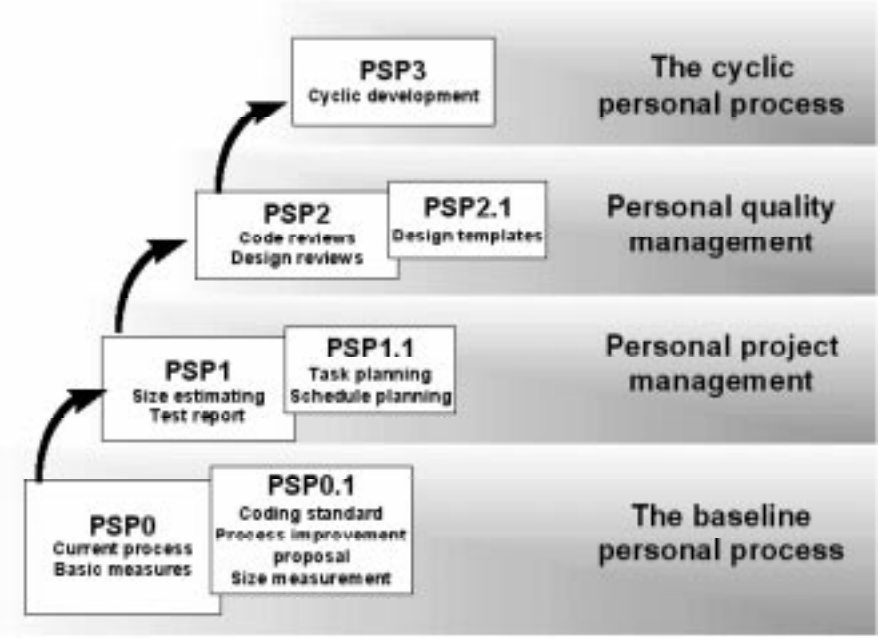


Figure 2-1: The PSP Process Levels

2.1.2 The Baseline Personal Process - PSP0 and PSP0.1

The baseline personal process (PSP0 and PSP0.1) provides an introduction to the PSP and establishes an initial base of historical size, time, and defect data. Engineers write three programs at this level. They are allowed to use their current methods, but do so within the framework of the six steps in the baseline process shown in Table 2-1.

PSP0 introduces basic process measurement and planning. Development time, defects, and program size are measured and recorded on provided forms. A simple plan summary form is used to document planned and actual results. A form for recording process improvement proposals (PIPs) is also introduced (PSP0.1). The PIP form provides engineers with a convenient way to record process problems and proposed solutions.

Step	Phase	Description
1	Plan	Plan the work and document the plan
2	Design	Design the program
3	Code	Implement the design
4	Compile	Compile the program and fix and log all defects found
5	Test	Test the program and fix and log all defects found
6	Postmortem	Record actual time, defect, and size data on the plan

Table 2-1: Steps in the Baseline PSP

2.1.3 Personal Project Management - PSP1 and PSP1.1

PSP1 and PSP1.1 focus on personal project management techniques, introducing size and effort estimating, schedule planning, and schedule tracking methods. Size and effort estimates are made using the PROBE method. (PROBE stands for PRoxy-Based Estimating.) With PROBE, engineers use the relative size of a proxy to make their initial estimate, then use historical data to convert the relative size of the proxy to LOC. Example proxies for estimating program size are objects,[2] functions, and procedures. For object-oriented languages, the relative size of objects and their methods is used as a proxy. For procedural languages, the relative size of functions or procedures is used as a proxy. Any proxy for size may be used so long as the proxy is correlated with effort, can be estimated during planning, and can be counted in the product. Other examples include screens or screen objects, scripts, reports, and document pages.

Using PROBE, the size estimate is made by first identifying all of the objects that must be developed. Then the type and relative size of the object are determined. The type refers to the general category of component—e.g., computational, input/output, control logic, etc. The five relative size ranges in the PSP are: very small, small, medium, large, and very large. The relative size is then converted to LOC using a size range table based on historical size data for the proxy. The estimated size of the newly developed code is the sum of all new objects, plus any modifications or additions to existing base code. Predicted program size and effort are estimated using the statistical method linear regression. Linear regression makes use of

2. In the textbook *A Discipline for Software Engineering* [Humphrey 95], the word object is often used as a synonym for class, function, or procedure.

the historical relationship between prior estimates of size and actual size and effort to generate predicted values for program size and effort. Finally, a prediction interval is calculated that gives the likely range around the estimate, based on the variance found in the historical data. The prediction interval can be used to assess the quality of the estimate.

PSP uses the earned value method for schedule planning and tracking. The earned value method is a standard management technique that assigns a planned value to each task in a project. A task's planned value is based on the percentage of the total planned project effort that the task will take. As tasks are completed, the task's planned value becomes earned value for the project. The project's earned value then becomes an indicator of the percentage of completed work. When tracked week by week, the project's earned value can be compared to its planned value to determine status, to estimate rate of progress, and to project the completion date for the project.

2.1.4 Personal Quality Management - PSP2 and PSP2.1

PSP2 and PSP2.1 add quality management methods to the PSP: personal design and code reviews, a design notation, design templates, design verification techniques, and measures for managing process and product quality.

The goal of quality management in the PSP is to find and remove all defects before the first compile. The measure associated with this goal is yield. Yield is defined as the percent of defects injected before compile that were removed before compile. A yield of 100% occurs when all the defects injected before compile are removed before compile.

Two new process steps, design review and code review, are included at PSP2 to help engineers achieve 100% yield. These are personal reviews conducted by an engineer on his/her own design or code. They are structured, data-driven review processes that are guided by personal review checklists derived from the engineer's historical defect data.

Starting with PSP2, engineers also begin using the historical data to plan for quality and control quality during development. Their goal is to remove all the defects they inject before the first compile. During planning, they estimate the number of defects that they will inject and remove in each phase. Then they use the historical correlation between review rates and yield to plan effective and efficient reviews. During development, they control quality by monitoring the actual defects injected and removed versus planned, and by comparing actual review rates to established limits (e.g., less than 200 lines of code reviewed per hour). With sufficient data and practice, engineers are capable of eliminating 60% to 70% of the defects they inject before their first compile.

Reviews are quite effective for eliminating most of the defects found in compile, and many of the defects found in test. But to substantially reduce test defects, better quality designs are needed. PSP2.1 addresses this need by adding a design notation, four design templates, and design verification methods to the PSP. The intent is not to introduce a new design method, but to ensure that the designer examines and documents the design from different

perspectives. This improves the design process and makes design verification and review more effective. The design templates in the PSP provide four perspectives on the design: an operational specification, a functional specification, a state specification, and a logic specification.

2.1.5 Cyclic Personal Process - PSP3

The Cyclic Personal Process, PSP3, addresses the need to efficiently scale the PSP up to larger projects without sacrificing quality or productivity. In the class engineers learn that their productivity is highest between some minimum and maximum size range. Below this range, productivity declines due to fixed overhead costs. Above this range, productivity declines because the process scalability limit has been reached. PSP3 addresses this scalability limit by introducing a cyclic development strategy where large programs are decomposed into parts for development and then integrated. This strategy ensures that engineers are working at their maximum productivity and product quality levels, with only incremental, not exponential, increases in overhead for larger projects.

To support this development approach, PSP3 introduces high-level design, high-level design review, cycle planning, and development cycles based on the PSP2.1 process. Two new forms are also introduced: a cycle summary to summarize size, development time, and defects for each cycle; and an issue tracking log for documenting issues that may affect future or completed cycles. Using PSP3, engineers decompose their project into a series of PSP2.1 cycles, then integrate and test the output of each cycle. Because the programs they produce with PSP2.1 are of high quality, integration and test costs are minimized.

2.1.6 Course Structure and Assignments

The PSP textbook, *A Discipline for Software Engineering* [Humphrey 95], describes a standard PSP course structure. This structure, shown in Table 2-2, includes the course topics covered in the lecture and assigned reading, the associated programming exercise, its description, and the PSP process level used for the assignment.

Course Topic	Exercise	Exercise Description	PSP Level
The Personal Software Process Strategy and the Baseline PSP	1A	Calculate the mean and standard deviation of N real numbers stored in a linked list	PSP0
The Planning Process	2A	Count the LOC in a program source file	PSP0.1
	R1	Produce a LOC counting standard	
	R2	Produce a coding standard	
Measuring Software Size	3A	Enhance program 2A to count object LOC or function/procedure LOC	PSP0.1
	R3	Defect analysis report	
Estimating Software Size	4A	Calculate the linear regression parameters for N pairs of real numbers stored in a linked list	PSP1
Resource and Schedule Estimating	5A	Numerical integration using Simpson's rule	PSP1.1
Measurements in the Personal Software Process	6A	Enhance program 4A to calculate a 90% and 70% prediction interval	PSP1.1
Design and Code Reviews	R4	Midterm process analysis report	
Software Quality Management	7A	Calculate the correlation of N pairs of real numbers stored in a linked list	PSP2
Software Design	8A	Sort a linked list	PSP2 or PSP2.1
Software Design Verification	9A	Chi-square test for normality	PSP2.1
Scaling-up the PSP	10A	Calculate the multiple linear regression parameters for N sets of four real numbers stored in a linked list	PSP3
Defining the Software Process and Using the PSP	R5	Final process analysis report	

Table 2-2: Standard Course Structure

The exercise sequence shown in Table 2-2 is for the "A series" exercises. An alternate "B series" containing nine exercises is also provided in the textbook. In addition to the programming exercises, there are also five report exercises including two standards (R1 and R2), and three analyses (R3, R4, and R5).

The PSP course is a one-semester course in an academic setting. Several different course formats have been used in industry. A popular format consists of a one-week session on planning, covering PSP0 and PSP1, followed by a one-week session on quality that covers PSP2 and PSP3. Each session is separated by a month. Two to three homework assignments

are assigned after each training session. The PSP course has also been taught in one-day-per-week, two-day-per-week, and one-day-every-other week formats. So long as the standard course structure is used, and students are given sufficient time to complete the exercises, the course schedule can be varied without affecting results.

2.2 PSP Measures

2.2.1 Measurement Overview

This section provides an overview of the basic PSP measures, forms, and measurement processes. This information is provided to give the reader some context for data that were analyzed for this study.

There are three basic measures in the PSP: development time, defects, and size. All other PSP measures are derived from these three basic measures. The measurement process and forms for these measures are introduced during the first three assignments at PSP process levels PSP0 and PSP0.1. Development time and defect measures are introduced on the first assignment; size is deferred until a program for counting LOC has been developed in assignment 2.

Development Time Measurement

Minutes are the unit of measure for development time. Engineers track the number of minutes they spend in each PSP phase, less time for any interruptions such as phone calls, coffee breaks, etc. A form, the Time Recording Log, is used to record development time.

The example Time Recording Log (Figure 2-2) illustrates how this form is used. In the example, the engineer started the Plan phase of his project on May 13 at 7:58 and finished planning at 8:45. The elapsed time was 47 minutes, but actual effort, or Delta Time, was only 44 minutes, due to an interruption of three minutes to take a phone call. The engineer started the Design phase at 8:47 and finished at 10:29. A two-minute interruption gives a Delta Time of 100 minutes. The remaining phases, Code, Compile, and Test are recorded in a similar manner.

The advantages of this approach to measuring development time are

- Using minutes is precise and simplifies calculations involving development time.
- Recording interruptions to work reduces the number of time log entries, provides a more accurate measure of the actual time spent, and a more accurate basis for estimating actual development time.
- Tracking interruption time separately can help engineers deal objectively with issues that affect time management, such as a noisy work environment or inappropriate mix of responsibilities (e.g., software development and help desk support).
- Time log entries take substantially less than a minute to record, but provide a wealth of detailed historical data for planning, tracking, and process improvement.

Time Recording Log

Date	Start	Stop	Interruption Time	Delta Time	Phase	Comments
5/13	7:58	8:45	3	44	Plan	phone call
	8:47	10:29	2	100	Design	create and review design
	7:49	8:59		70	Code	coded main and all functions
	9:15	9:46		31	Compile	compiled and linked
	9:47	10:10		23	Test	ran tests A, B, and C
	4:33	4:51		18	Postmortem	

Figure 2-2: Time Recording Log

A defect is defined as any change that must be made to the design or code in order to get the program to compile or test correctly. Defects are recorded on the Defect Recording Log as they are found and fixed. The example Defect Recording Log (Figure 2-4) shows the information that is recorded for each defect: the date, sequence number, defect type, phase in which the defect was injected, phase in which it was removed, fix time,[3] and a description of the problem and fix.

When an engineer injects a new defect while trying to fix an existing defect, proper accounting of fix time becomes more complicated. A common mistake is to include the fix time for the new defect twice. To help with this problem, a space is provided to record a reference to the original defect that was being fixed. The number of the original defect that was being fixed is recorded in the fix defect reference of the new defect.

3. The time, in minutes, spent finding and fixing the defect.

Each defect is classified according to a defect type standard. The standard includes 10 defect types (see Figure 2-3) in a simple, easy-to-use classification scheme designed to support defect analysis. Engineers can refine the standard to meet personal needs, but they are encouraged to wait until they have sufficient data to justify a change.

In the example Defect Recording Log (Figure 2-4), the engineer found the first defect on May 13. The defect was a type 20 (syntax error) that was injected during the code phase and removed during compile. The engineer spent 22 minutes finding and fixing the defect. The second error, also a syntax error, was injected during the code phase and removed during compile, and took 18 minutes to find and fix.

Type Number	Type Name	Description
10	Documentation	comments, messages
20	Syntax	spelling, punctuation, typos, instruction formats
30	Build, Package	change management, library, version control
40	Assignment	declaration, duplicate names, scope, limits
50	Interface	procedure calls and references, I/O, user formats
60	Checking	error messages, inadequate checks
70	Data	structure, content
80	Function	logic, pointers, loops, recursion, computation, function defects
90	System	configuration, timing, memory
100	Environment	design, compile, test, or other support system problems

Figure 2-3: Defect Type Standard

Defect Recording Log

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	**1**	**20**	**CODE**	**CMPL**	**22**	

Description: **syntax error in scanf statement**

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	**2**	**20**	**CODE**	**CMPL**	**18**	

Description: **error in linked list struct type declarations within access functions**

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	**3-6**	**20**	**CODE**	**CMPL**	**1**	

Description: **missing ;**

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	**7**	**20**	**CODE**	**CMPL**	**1**	

Description: **incorrect spelling of identifier in declaration**

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	**8**	**20**	**CODE**	**CMPL**	**1**	

Description: **function declaration error**

Date	Number	Type	Inject	Remove	Fix Time	Fix Defect
5/13	**9**	**30**	**CODE**	**TEST**	**1**	

Description: **link error, missing include for math.h**

Figure 2-4: Defect Recording Log

Size Measurement

The primary purpose of size measurement in the PSP is to provide a basis for estimating development time. Lines of code were chosen for this purpose because they meet the following criteria: they can be automatically counted, precisely defined, and are well correlated with development effort based on the PSP research [Humphrey 95, pp. 115-116]. Size is also used to normalize other data, such as productivity (LOC per hour) and defect density (defects per KLOC). While LOC are suitable for the programming assignments in the PSP course, any measure that meets these same criteria can be used in practice.

In the PSP course, as in practice, each program involves some amount of new development, enhancement, and/or reuse. Therefore, the total LOC in a program will have several different sources, including some new LOC, some existing LOC that may have been modified, and some reused LOC. Because LOC are the basis for estimates of development time, it is important to account for these different types of LOC separately.

PSP uses the LOC accounting scheme shown in Table 2-3. Base LOC are any LOC from an existing program that will serve as the starting point for the program being developed. Deleted and modified LOC are those base LOC that are being deleted or modified. Added LOC is the sum of all newly developed object, function, or procedure LOC, plus additions to the base LOC. Reused LOC are the LOC taken from the engineer's reuse library and used without modification. If these LOC are modified, then they are considered to be base LOC.

New and changed LOC is the sum of added LOC and modified LOC. New and changed LOC, not total LOC, is the most commonly used size measure in the PSP. For example, new and changed LOC are the basis for size and effort estimating, productivity (LOC/hour), and defect density (defects/KLOC). Please note that improvements in quality and productivity found in this study are therefore not the result of increased reuse. Finally, total LOC is the total program size, and total new reused LOC are those added LOC that were written to be reused in the future.

Type of LOC	Definition
Base	LOC from a previous version
Deleted	Deletions from the Base LOC
Modified	Modifications to the Base LOC
Added	New objects, functions, procedures, or any other added LOC
Reused	LOC from a previous program that is used without modification
New & Changed	The sum of Added and Modified LOC
Total LOC	The total program LOC
Total New Reused	New or added LOC that were written to be reusable

Table 2-3: PSP LOC Type Definitions

Program Size (LOC):	Plan	Actual	To Date
Base(B)	0	0	
	(Measured)	(Measured)	
Deleted (D)	0	0	
	(Estimated)	(Counted)	
Modified (M)	0	0	
	(Estimated)	(Counted)	
Added (A)	112	137	
	(N-M)	(T-B+D-R)	
Reused (R)	174	174	316
	(Estimated)	(Counted)	
Total New & Changed (N)	112	137	759
	(Estimated)	(A+M)	
Total LOC (T)	286	311	1161
	(N+B-M-D+R)	(Measured)	
Total New Reused	0	0	326

Time in Phase (min.)	Plan	Actual	To Date	To Date %
Planning	42	44	287	17.2
Design	76	56	490	29.4
Design review				
Code	48	61	337	20.2
Code review	30	27	27	1.6
Compile	15	1	106	6.4
Test	26	38	326	19.6
Postmortem	13	20	94	5.6
Total	250	247	1667	100.0

Defects Injected	Plan	Actual	To Date	To Date %
Planning	0	0	0	0.0
Design	2	1	10	18.2
Design review	0	0	0	0.0
Code	6	6	40	72.7
Code review	0	0	0	0.0
Compile	0	0	2	3.6
Test	0	0	3	5.5
Total Development	8	7	55	100

Defects Removed	Plan	Actual	To Date	To Date %
Planning	0	0	0	0.0
Design	0	0	0	0.0
Design review	0	0	0	0.0
Code	0	0	1	1.8
Code review	0	5	5	9.1
Compile	4	0	24	43.6
Test	4	2	25	45.5
Total Development	8	7	55	100.0
After Development	0	0	0	

Figure 2-5: Sample PSP Project Plan Summary Form

Project Summary Data

Project summary data are recorded on the Project Plan Summary form. This form provides a convenient summary of planned and actual values for program size, development time, and defects, and a summary of these same data for all projects completed to date. The project plan summary is the source for all data used in this study. Figure 2-5 shows the four sections of the project plan summary that were used: Program Size, Time in Phase, Defects Injected, and Defects Removed.

The data on the plan summary form has many practical applications for the software engineer. The data can be used to track the current project, as historical data for planning future projects, and as baseline process data for evaluating process improvements.

2.2.2 PSP Derived Measures

Each PSP level introduces new measures to help engineers manage and improve their performance. These measures are derived from the three basic PSP measures: development time, defects, and size.

Table 2-4 contains a partial list of the derived measures available in the PSP and their definitions. These measurement definitions are included to provide context for the analyses that follow.

Measure	Definition
Interruption time	The elapsed time for small interruptions from project work such as a phone call
Delta Time	Elapsed time in minutes from start to stop less interruptions Stop - Start - Interruptions
Planned Time in Phase	The estimated time to be spent in a phase for a project
Actual Time in Phase	The sum of Delta Times for a phase of a project
Total Time	The sum of planned or actual time for all phases of a project
Time in Phase To Date	The sum of Actual Time in Phase for all completed projects
Total Time To Date	The sum of Time in Phase To Date for all phases of all projects
Time in Phase To Date%	100 * Time in Phase To Date for a phase divided by Total Time in Phase To Date
Compile Time	The time from the start of the first compile until the first clean compile
Test Time	The time from the start of the initial test until test completion
Defect	Any element of a program design or implementation that must be changed to correct the program[a]
Defect type	See Figure 2-3, Defect Type Standard
Fix Time	The time to find and fix a defect

Table 2-4: Definitions of PSP Measures

Measure	Definition
LOC	A logical line of code as defined in the engineers counting and coding standard
LOC Type	See Table 2-2, LOC Types
LOC/Hour	Total new and changed LOC developed divided by the total development hours
Estimating Accuracy	The degree to which the estimate matches the result. Calculated for time and size %Error = 100*(Actual-Estimate)/Estimate
Test Defects/KLOC	The defects removed in the test phase per new and changed KLOC 1000*(Defects removed in Test)/Actual New and Changed LOC
Compile Defects/KLOC	The defects removed in compile per new and changed KLOC 1000*(Defects removed in Compile)/Actual New and Changed LOC
Total Defects/KLOC	The total defects removed per new and changed KLOC 1000*(Total Defects removed)/Actual New and Changed LOC
Yield	The percent of defects injected before the first compile that are removed before the first compile 100*(defects found before the first compile)/(defects injected before the first compile)
Appraisal Time	Time spent in design and code reviews
Failure Time	Time spent in compile and test
Cost of Quality (COQ)	Cost of Quality = Appraisal Time + Failure Time[b]
COQ Appraisal/Failure Ratio (A/FR)	A/FR = Appraisal Time/Failure Time
Review Rate	Review rate is lines of code reviewed per hour[c] 60 * New and Changed LOC/review minutes

Table 2-4: Definitions of PSP Measures

a. An error or mistake made by a software engineer becomes a defect when it goes undetected during design or implementation. If it is corrected before the end of the phase in which it was injected then typically there is no defect. If it is found at the phase-end review or during compile, test, or after test, then a defect is recorded.

b. The standard definition of COQ includes appraisal cost, failure cost, and prevention cost. Only appraisal costs and failure costs are included in the PSP.

c. Design review rates are based on new and changed LOC. For planning purposes, engineers use estimated new and changed LOC to arrive at planned review rates.

3 Overview of the Data Set and Statistical Model

This section provides a brief summary of the data used in this study, as well as an overview of the statistical model used. More complete information on these two topics is provided in Appendices A and B.

3.1 The Data Set

Each programming assignment results in some 70 pieces of data being collected by each engineer. These data are used by the engineers to monitor their work on the individual assignments, as well as to analyze their personal software process for improvement decisions. The PSP training course also identifies specific data analyses for engineers to perform and submit as part of the reports written during the class.

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 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. However, our secondary analysis suffers from problems associated with using data intended for another purpose. That is, the data were collected to provide feedback to individual engineers, not for us to write this report. There are many cases where instructors tailored the training course (including selection of assignments, data collection requirements, and sequence of introduction for process changes). In addition, there are cases where engineers did not complete the entire course.

In all, a total of 23 PSP classes consisting of 298 engineers provided the data used in this report. Each analysis presented is based on at least 170 cases where complete data were avaialble for that analysis. Detailed information about the data set and selection criteria for inclusion in each analysis are provided in Appendix A. To summarize the data in gross terms, over 300,000 lines of code were written during a total of more than 15,000 hours by 298 engineers. During this time, the engineers discovered and removed approximately 22,000 defects.

3.2 Statistical Model

Engineers use statistical methods during PSP training to analyze their personal data and make judgments about the benefits of changes to their personal software processes. In this report, we perform a secondary analysis of these engineers' data in aggregate form. The benefits associated with a larger pool of data are well understood in research of this type. Unless there is evidence of general applicability, the stellar performance of an individual engineer provides little incentive for widespread use of any methodology.

Differences in performance between engineers is typically the greatest source of variability in software engineering research, and this study is no exception. However, the design of the PSP training class, and the standardization of each engineer's measurement practice, allow the use of statistical models which are well suited for dealing with the variation among engineers.

In the analyses summarized here, the changes in engineers' data over the course of nine programming assignments are studied. Rather than analyzing changes in group averages, this study focuses on the average changes of individual engineers. Some engineers performed better than others from the first assignment, and some improved faster than others during the course of training. In order to discover the pattern of improvement in the presence of these natural differences between engineers, the statistical method known as the repeated measures analysis of variance (ANOVA) is used [Tabachnick 89]. This method is briefly described below; a more detailed explanation of this method is provided in Appendix B.

In brief, the repeated measures analysis of variance takes advantage of situations where the same people are measured over a succession of trials. By treating previous trials as baselines, the differences in measures across trials (rather than the measures themselves) are analyzed to uncover trends across the data. This allows for differences among baselines to be factored out of the analysis. In addition, the different rates of improvement between people can be viewed more clearly. If the majority of people change substantially (relative to their own baselines), the statistical test will reveal this pattern. If only a few people improve in performance, the statistical test is not likely to suggest a statistically significant difference, no matter how large the improvement of these few people.

In the following five sections, repeated measures analysis of variance is used to test hypotheses about the intended benefits of PSP training. Each analysis is focused on a subset of the measures collected, as appropriate to the hypothesis at hand. Selection criteria are used to combat problems associated with missing data, as well as data entry errors. These selection criteria are fully described in Appendix A.

The focus of the hypotheses tested, and therefore the focus of the statistical analyses, is on changes across the first three major process levels in the training class. Each level (PSP0, PSP1, and PSP2) contains three programming assignments. The data from the three assignments are pooled together (as described in Appendix A) to form process-level metrics for each engineer. Therefore, in the analysis, the three assignments in each level are treated as three instantiations of the same process. Strictly speaking, there are process changes that occur during these three assignments. However, the hypotheses tested are specific to the process changes that occur among the major PSP levels. Therefore, the more minor process changes that occur from assignment to assignment within a process level are not studied here.

Support for the appropriateness of this treatment is obtained when the statistical model is altered to include an “assignment within level” effect.[1] In this model, statistical tests are carried out to establish whether or not the three assignments within PSP levels differ from each other, as well as statistical tests for differences between PSP levels. Using this more elaborate model, the differences between PSP levels identified by the analyses presented in Sections 4 through 8 were all found to be statistically significant. These statistical findings indicate that the differences between PSP levels remain, even when systematic differences between the assignments within levels is factored out.

For each of the five analyses, when the statistical test revealed a significant difference among the three PSP levels, follow-up analyses were performed to compare two PSP levels at a time. These follow-up analyses helped to more clearly identify where changes occur in the course of training. See Appendix B for a more complete (and technically complex) discussion of the statistical procedures.

1. This effect represents the differences among the first, second, and third assignments in the PSP levels. The comparison is among (1,4,7), (2,5,8), and (3,6,9). The analysis reported here does not compare individual programming assignments.

4 Size Estimation

Estimating the size of a job prior to deciding how long it will take to complete, while logical to most people, seems to be a difficult practice to instill in software engineers. The PSP provides a proxy-based estimation method (introduced during PSP level 1) to help engineers decompose the program and estimate the size of each element, based on historical data. Introduction of this method is designed to enable engineers to become more accurate estimators of their own work.

While there will always be a subjective element to estimation no matter how much data it is based on, the PSP training strives to teach engineers how to make the best use of their own past experience. When the size estimation method is introduced at the start of PSP level 1, the engineers have data from the three previous assignments as a basis for estimating the fourth. Therefore, the hypothesis tested in this section of the study is as follows:

> *As engineers progress through the PSP training, their size estimates gradually grow closer to the actual size of the program at the end. More specifically, with the introduction of a formal estimation technique for size in PSP level 1, there is a notable improvement in the accuracy of engineers' size estimates.*

4.1 Group Trend

The distributions shown in Figure 4-1 illustrate the performance of engineers' size estimation for the three PSP levels. The values plotted were derived by summing the data[1] for the three assignments in each PSP level and computing the estimation accuracy value by calculating (Estimated Size - Actual Size) / Estimated Size.[2] With the exception of assignment 1 (where size estimates are not required), only the data for engineers who provided size estimates for all programs are included in the charts and the subsequent analysis. In all, 170 engineers are represented here.

The horizontal axis on each of the three plots in Figure 4-1 represents the value of estimation accuracy, ranging from -300% to +100%.[3] The vertical axis represents the number of engineers at a particular level of estimation accuracy.

---

1. All representations of program size (whether estimated or actual) are based on new and changed lines of code.

2. This formula differs from the one used in the PSP training class. For the purpose of this study the Actual is subtracted from the Estimate so that underestimates result in a negative value and overestimates result in a positive value. In the training class the equation used is (Actual - Estimate)/Estimate.

3. As one might infer from this range of values for estimation accuracy, the computation of this metric leads to a skewed distribution by definition. This matter is discussed in detail in Appendix B.

Improvement in accuracy for any given engineer could be seen in two different ways. First, the absolute distance of the estimation accuracy metric from zero would be reduced. Second, over the course of several assignments, both overestimates and underestimates would be seen. This is in contrast to a pattern of consistent underestimating (or overestimating).

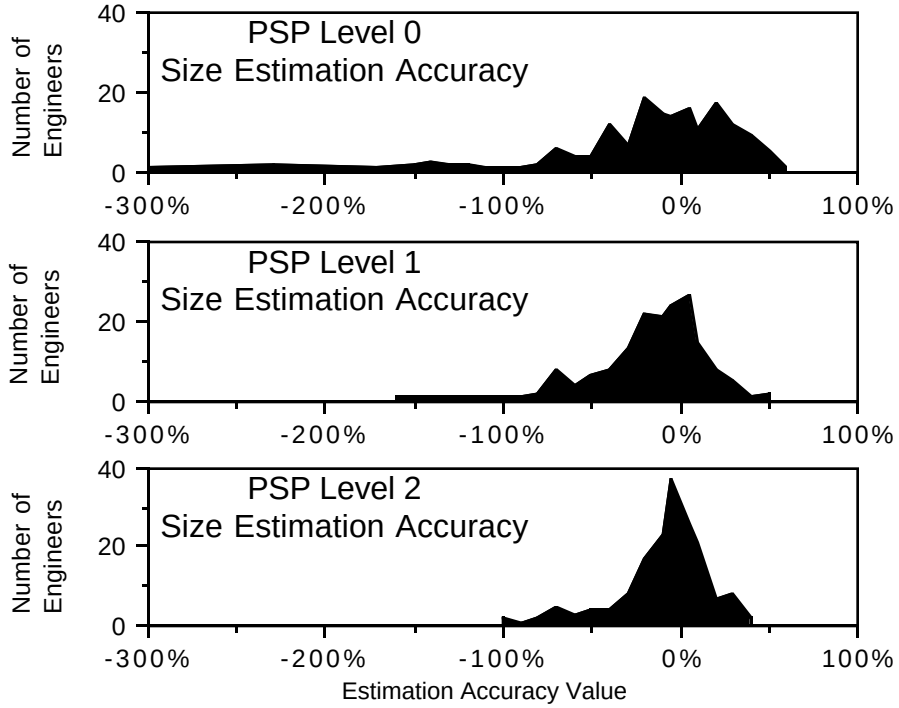


Figure 4-1: Distributions of Size Estimation Accuracy by PSP Level

With respect to the performance of the group, the distributions in Figure 4-1 show a general reduction in the distance from zero for the values of size estimation accuracy. The long 'tail' of the distribution in the top panel for PSP 0 (stretching out to -300%) is considerably shortened by the third panel, which represents PSP level 2. In addition, the distribution appears to be more symmetrically spread around zero in the third panel. Finally, the number of engineers achieving a value near zero during PSP level 2 is more than twice that of PSP level 0.

4.2 Analysis of Individual Changes in Size Estimation Accuracy

While the trends illustrated above support the claim of improved size estimation for the group, the goal of PSP is to help individual engineers. The repeated measures ANOVA conducted shows that on average, individual engineers do improve their size estimation accuracy.

The analysis comparing the pooled size estimation accuracy values for each PSP level for each engineer revealed statistically significant differences across the three PSP levels ($p = .041$). The follow-up analysis comparing adjacent pairs of PSP levels revealed statistically significant differences between PSP levels 0 and 1 ($p = .037$). The difference between PSP levels 1 and 2 is not statistically significant.

These statistical findings support the hypothesis that size estimation improves with PSP, and that the improvement is most notable when the PROBE estimation method is introduced during PSP level 1.

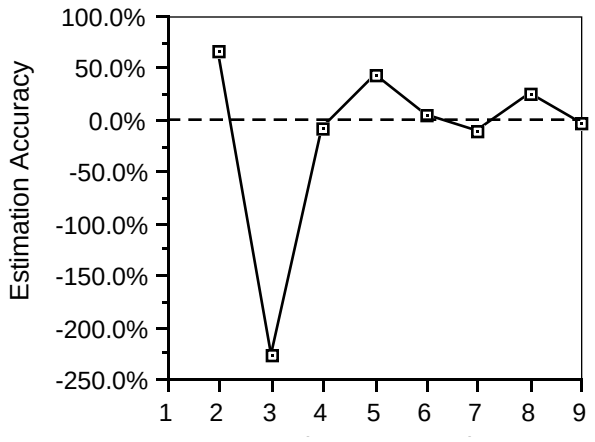
4.3 Summary of Improvements in Size Estimation Accuracy

Accurate size estimates are a fundamental building block for realistic project plans. Training in PSP provides individual engineers with an ability to improve their skill in estimating the size of the products they produce. This ability is clearly demonstrated in the results presented here.

The median individual improvement in size estimation accuracy is a factor of 2.5. This value derives from examining the ratio of size estimation accuracy during PSP level 0 and size estimation accuracy during PSP level 2. Hence, 50% of the engineers reduced the error in their size estimates by a factor of 2.5 or greater.

To illustrate this improvement in more concrete terms, the table below reflects the performance of a single engineer selected from the data set. This example shows one of the more outstanding improvements observed in the data set.

Assignment	Estimated Size	Actual Size	Size Estimation Accuracy	Trend in Size Estimation Accuracy
1	N/A	67	N/A	
2	100	34	66%	
3	60	197	-228%	
4	106	113	-7%	
5	87	49	44%	
6	155	149	4%	
7	137	152	-11%	
8	97	72	26%	
9	268	280	-4%	



PSP0 aggregate size estimation accuracy = - 44.38%

PSP2 aggregate size estimation accuracy = - 0.40%

Size estimation accuracy improvement factor = 111.4

Table 4-1: Sample Data for Size Estimation

5 Effort Estimation

Hardly a single software engineer has escaped the need to estimate how long it would take them to produce some product—be it a module of code, a design specification, or a series of test cases. However, the question posed to the practicing engineer typically takes the form of "Can you have it done by next week?" rather than a request for an honest evaluation of how much effort is required. The intuition-based algorithms in the mind of most seasoned software engineers are typically the result of years of experience, putting their reputation and credibility on the line with supervisors.

The PSP helps engineers in this environment by arming them with effort estimation data. In this section, data are used to test the following hypothesis:

> *As engineers progress through the PSP training, their effort estimates grow closer to the actual effort expended for the entire life cycle. More specifically, with the introduction of a statistical technique (linear regression) in PSP level 1, there is a notable improvement in the accuracy of engineers' effort estimates.*

5.1 Group Trend

The distributions shown in Figure 5-1 illustrate the performance of engineers' effort estimation for the three PSP levels. Again, the values plotted are based on pooling the effort estimates and the effort actuals for the three assignments in each level.

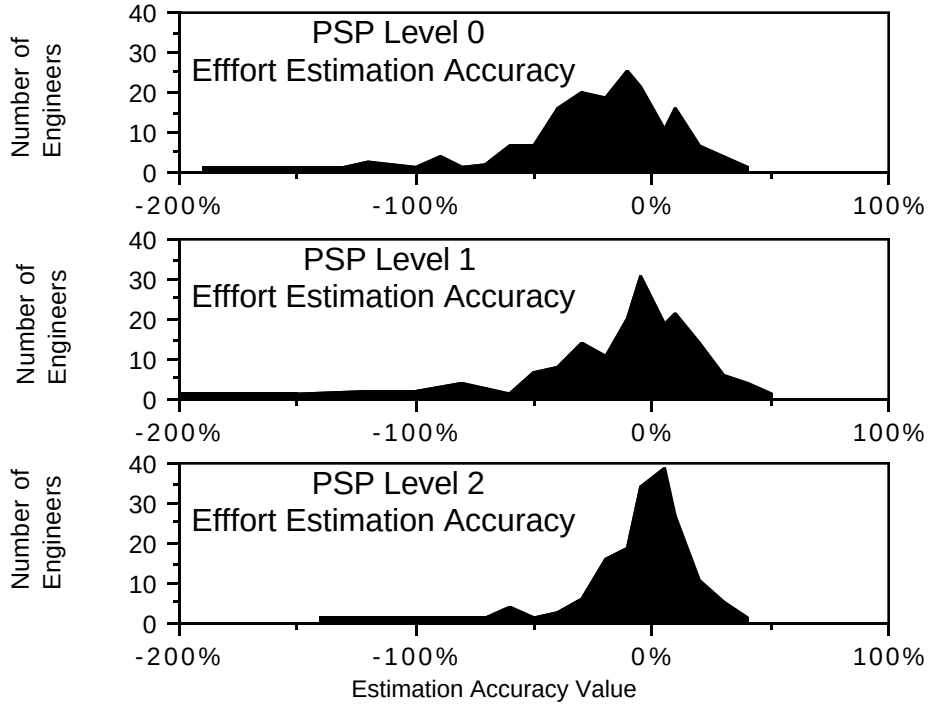


Figure 5-1: Distributions of Effort Estimation Accuracy by PSP Level

A comparison of the three distributions in Figure 5-1 shows that the majority of engineers underestimated their effort in PSP level 0. For PSP level 1, the distribution is more nearly symmetrical (the number of engineers overestimating is closer to the number of engineers underestimating), though underestimates of 200% still occur. Finally, for PSP level 2, the distribution is closer still to the desired shape (symmetrical with narrow range, centered on zero), and the number of engineers with estimation accuracy values near zero is substantially larger.

5.2 Analysis of Individual Changes in Effort Estimation Accuracy

The repeated measures ANOVA for effort estimation conducted shows that on average, individual engineers do improve their effort estimation accuracy.

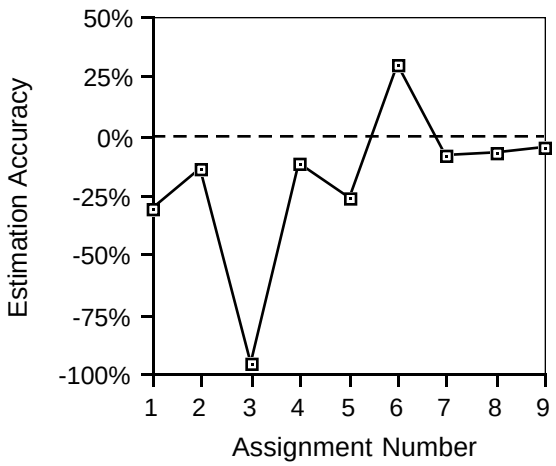
The analysis comparing the pooled effort estimation accuracy values for each PSP level revealed a statistically significant difference across the three PSP levels ($p < .0005$). The follow-up analysis comparing adjacent pairs of PSP levels revealed statistically significant differences between PSP levels 0 and 1 ($p < .0005$). The difference between PSP levels 1 and 2 is not statistically significant. These statistical findings support the hypothesis that effort estimation accuracy improves with PSP and that the improvement is most notable at PSP level 1.

5.3 Summary of Improvements in Effort Estimation Accuracy

Use of historical data for deriving effort estimates is common practice in the software industry today. However, estimation at the level of an individual engineer's workload remains a challenge. The PSP training provides engineers with the ability to make estimates, and to improve the estimating *process*, at the level of an individual engineer. This ability is clearly demonstrated in the results presented here.

The median improvement in effort estimation accuracy is a factor of 1.75. This value derives from examining the ratio of effort estimation accuracy during PSP level 0 and effort estimation accuracy during PSP level 2. Hence, 50% of the engineers reduced the error in their effort estimates by a factor of 1.75 or greater.

Again, to illustrate this improvement in more concrete terms, the table below reflects the performance of a single engineer selected from the data set.

Assignment	Estimated Effort	Actual Effort	Effort Estimation Accuracy	Trend in Effort Estimation Accuracy
1	240	314	- 31%	
2	180	205	- 14%	
3	160	312	- 95%	
4	252	282	- 12%	
5	300	378	- 26%	
6	600	419	30%	
7	290	313	- 8%	
8	180	192	- 7%	
9	510	535	- 5%	

PSP0 aggregate effort estimation accuracy = - 43.28%

PSP2 aggregate effort estimation accuracy = - 6.12%

Effort estimation accuracy improvement factor = 7.1

Table 5-1: Sample Data for Effort Estimation

6 Defect Density

Defect counts and measures of defect density (i.e., defects per KLOC) have traditionally served as software quality measures. The PSP uses this method of measuring product quality, as well as several process quality metrics. The consequence of high defect density in software engineering is typically seen in the form of bug-fixing or rework effort incurred on projects.

In this section, the impact of PSP training on the defect density of the programs produced by the engineers is addressed. In addition to overall defect density, a specific focus on the defect density of programs during the compile and test phases of the life cycle is provided. Defects that remain in the product at the end of the life cycle are the most costly to remove and have frequently been used to estimate the defect density of the delivered product; therefore, a reduction in these 'late' defects has a beneficial effect above and beyond the impact of a reduction in overall defect density.

The hypotheses to be investigated in this section are as follows:

> *As engineers progress through PSP training, the number of defects injected and therefore removed per thousand lines of code (KLOC) decreases.*
>
> *With the introduction of design and code reviews in PSP level 2, the defect densities of programs entering the compile and test phases decrease significantly.*

6.1 Group Trend

Figure 6-1 depicts the change in average defects per KLOC removed during compile and test, as well as defects per KLOC over the entire life cycle. Data from 181 engineers who provided complete data were used in the figures and analyses in this section.

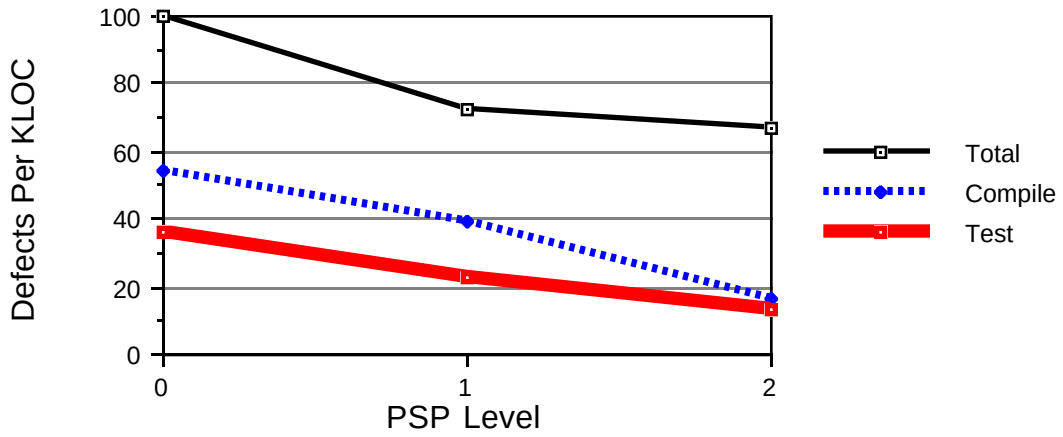


Figure 6-1: Trends in Average Defect Density

As Figure 6-1 shows, there is a modest decline in overall defect density across the three PSP levels. More importantly, the number of defects removed in the compile and test phases (taken together) are substantially lower for PSP level 2, as compared to PSP level 0.

Figure 6-2 (below) provides a more detailed examination of changes in defect density. The horizontal axis on each chart represents the number of defects per KLOC and the vertical axis represents the number of engineers reporting defect densities at a given value. The data plotted are pooled defect densities for each PSP level. The three distributions in the left column show defect densities in the compile phase, and the three distributions in the right column show the defect densities in the test phase.

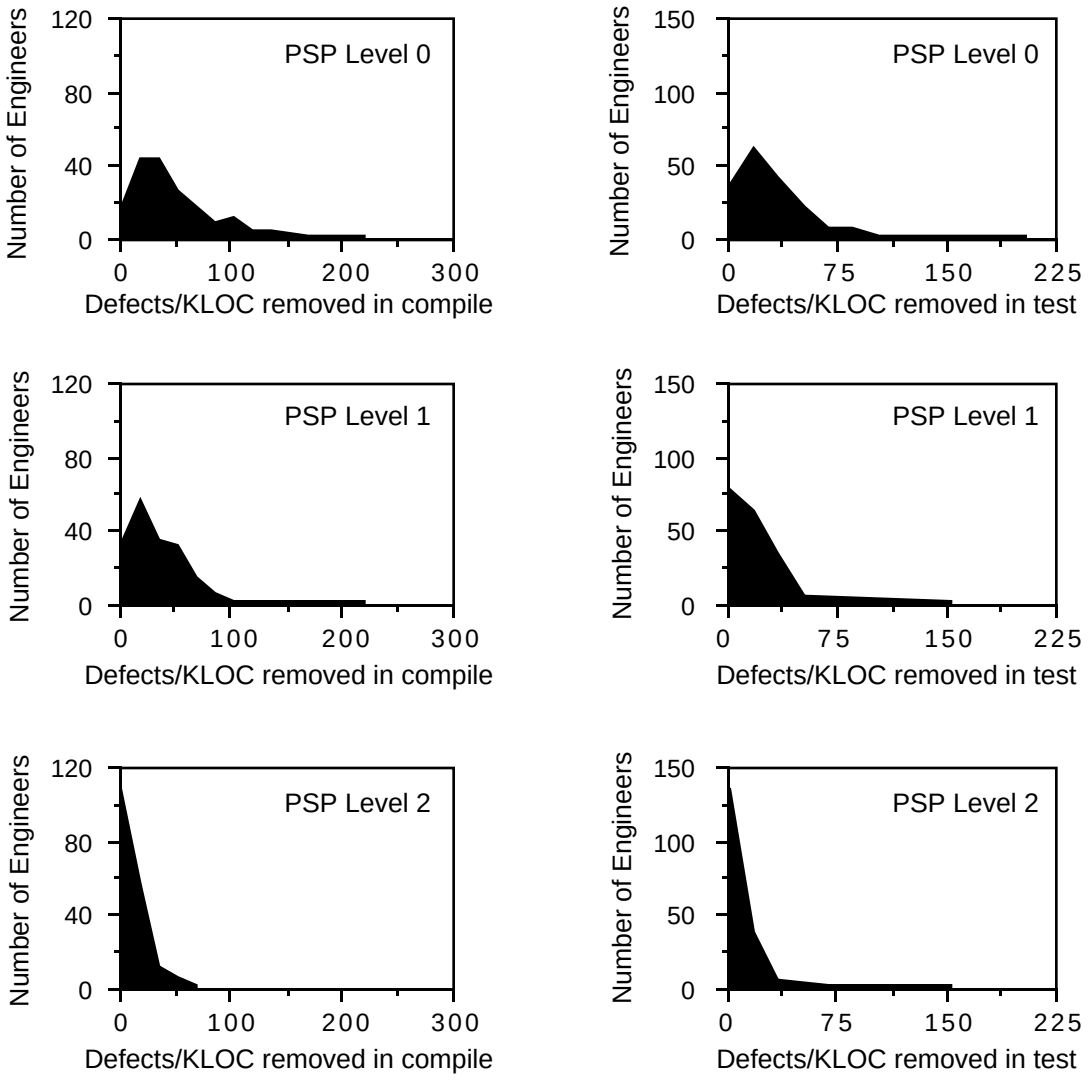


Figure 6-2: Defect Density Distributions for Compile and Test Phases

As Figure 6-2 illustrates, the defect densities in both the compile and test phases changed substantially during the course of PSP training. As the distributions of defect density in compile reveal, the vast majority of engineers reported fewer than 50 defects per KLOC during PSP level 2, compared to PSP level 0, when many engineers reported more than 50 defects per KLOC in compile. For defect density in the test phase, a similar decline in defects per KLOC is observed, this time with the majority of engineers reporting fewer than 25 defects per KLOC in test during PSP level 2.

6.2 Analysis of Individual Changes in Defect Density

6.2.1 Changes in Overall Defect Density

The analysis of overall defect density reveals a statistically significant difference in total defect density across the three PSP levels ($P < .0005$). Specific comparisons of the three PSP levels show that the difference in overall defect density between PSP levels 0 and 1 is statistically significant ($p < .0005$), but that the difference between PSP levels 1 and 2 is not.

6.2.2 Changes in Defect Density in the Compile Phase

The analysis of defects per KLOC for products entering the compile phase reveals a statistically significant difference in defect density across the three PSP levels ($p < .0005$). Specific comparisons of the three PSP levels also reveal statistically significant differences between PSP levels 0 and 1, as well as between PSP levels 1 and 2 ($p < .0005$ for both).

6.2.3 Changes in Defect Density in the Test Phase

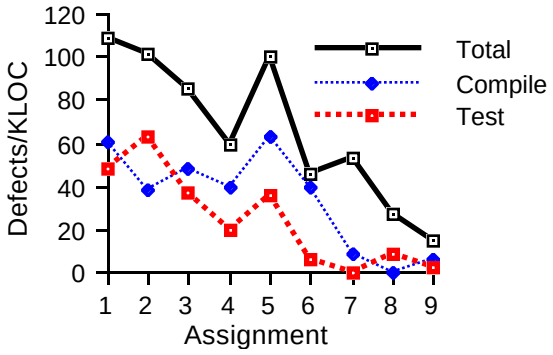
The analysis of defects per KLOC for products entering the test phase reveals a statistically significant difference in defect density across the three PSP levels ($p < .0005$). Specific comparisons of the three PSP levels also reveal statistically significant differences between PSP levels 0 and 1, as well as between PSP levels 1 and 2 ($p < .0005$ for both).

6.3 Summary of Improvements in Defect Density

A reduction in total defect density translates directly to a reduction in the amount of rework needed to field a product. Furthermore, as the burden of defect removal activity is shifted to earlier phases in the product life cycle, the cost of rework is significantly reduced.

The median reduction in total defect density is a factor of 1.5. The median reduction in defect density for the compile phase is a factor of 3.7, and for the test phase, the median reduction is a factor of 2.5. These quality improvements are illustrated in the table below with data from a single engineer.

Assignment	Total Defects per KLOC	Defects per KLOC Removed in Compile	Defects per KLOC Removed in Test	Trends in Defect Density Reduction
1	108.43	60.24	48.19	
2	101.27	37.97	63.29	
3	85.11	47.87	37.23	
4	58.82	39.22	19.61	
5	100.00	63.64	36.36	
6	45.45	39.77	5.68	
7	53.57	8.93	0.00	
8	27.03	0.00	9.01	
9	15.34	6.13	3.07	



	PSP0 Defect Density	PSP2 Defect Density	Improvement Factor
Total Defect Density	94.29	25.50	3.7
Compile Phase	48.57	5.46	8.9
Test Phase	45.71	3.64	12.6

Table 6-1: Sample Data for Defect Density

7 Pre-Compile Defect Yield

One of the most powerful process metrics used in the PSP training is the pre-compile defect yield (hereafter referred to simply as *yield*). Yield is the percentage of defects injected before the compile phase that are removed before the first compile. The PSP training teaches engineers to examine process quality by quantifying the yield of their personal software process. By understanding how well their process works to prevent defects from "entering" the last phases of the process, engineers can see for themselves the benefit of changes they make to their processes.[1] In general, the goal is to work for a yield of 100%.

The hypothesis to be addressed in this section is as follows:

> *As engineers progress through the PSP training, their yield increases significantly. More specifically, the introduction of design review and code review following PSP level 1 has a significant impact on the value of engineers' yield.*

7.1 Group Trend

Figure 7-1 shows the trend in average yield across the three PSP levels, with the dotted lines representing ± 1 standard deviation. Data from 188 engineers reporting complete data were used in the figures and analysis in this section.

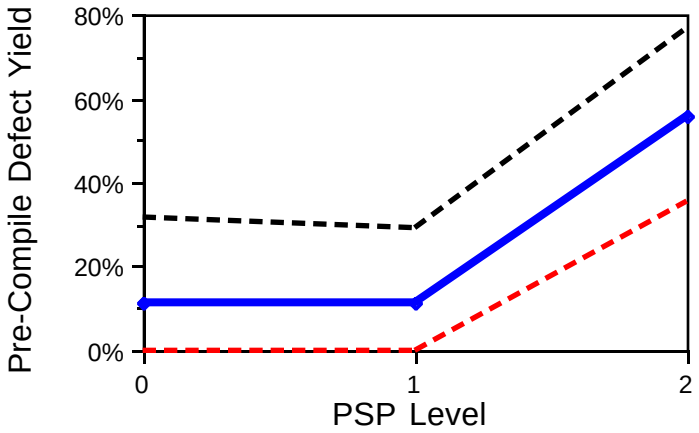


Figure 7-1: Average Yield

1. It should be noted here that PSP students are not encouraged to rely solely on their pre-compile defect yield as a quality indicator, and the data collected (including defect type, defect removal efficiency, and defect removal leverage) provide a more complete picture for diagnosis of personal software processes than this single analysis illustrates.

The increase in average yield from approximately 10% to nearly 55% from PSP level 1 to PSP level 2 is difficult to miss. The major process change that occurs between PSP levels 1 and 2 is the introduction of formal design reviews and code reviews.

A more detailed depiction of the changing values of yield is provided in the boxplots below. The rectangles in Figure 7-2 encapsulate the middle 50% of each distribution (with the lower edge of the "box" equal to the 25th percentile and the upper edge equal to the 75th percentile). The vertical line above each box extends to the highest value in each distribution that is not classified as an outlier.[2] The bar below the "box," similarly, extends to the lowest value that is not an outlier. The asterisk (*) in each box represents the median of the distribution (this is the value that divides the group evenly in half).

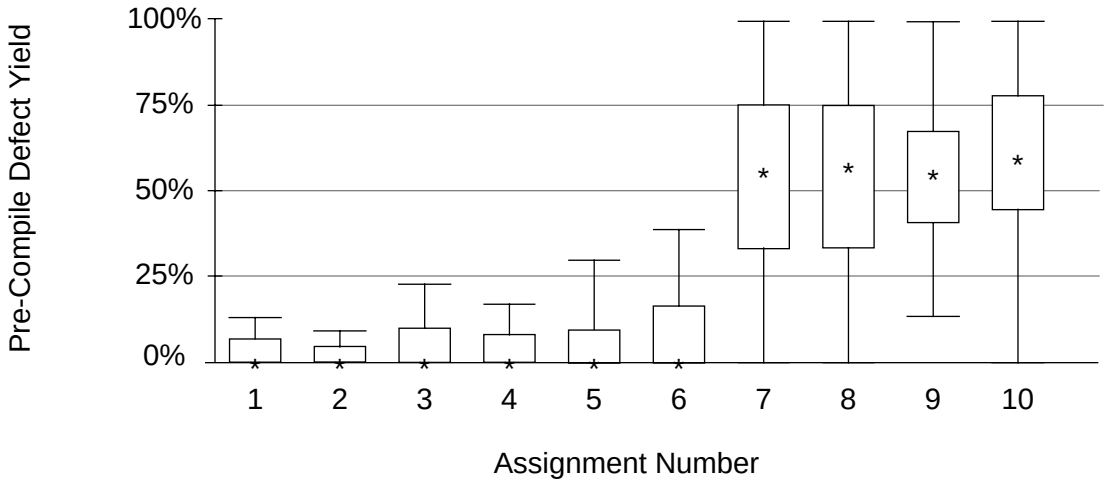


Figure 7-2: Yields for Each Assignment

As the figure illustrates, prior to assignment 7, very few engineers removed defects from their programs before the first compile. Starting with assignment 7, however, more than 75% of the engineers removed more than a quarter of the defects before they compiled the program.

By looking at the median for each assignment (the asterisk in each box) it is apparent that at the start of the training, half (or more) of the engineers removed no defects before compiling their programs. Starting with assignment 7, however, the majority of engineers removed more than half of the pre-compile defects before compilation.

2. There were many values of yield classified as outliers which are not shown in the figure above for the sake of clarity. The statistical analyses presented later in this section include all available data, including the outliers.

7.2 Analysis of Individual Changes in Yield

While the change in average yield for the group of engineers is apparent when looking at the figures discussed above, the changes in individual engineers' yields are what allow them to monitor their personal process.

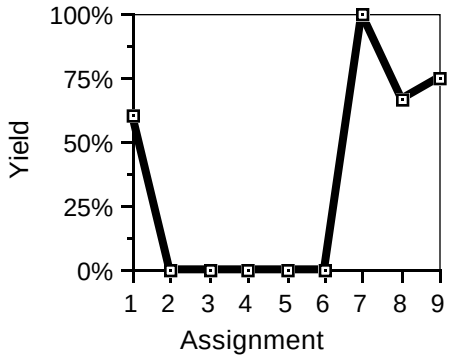
The repeated measures ANOVA reveals that the differences in yields across the three PSP levels is statistically significant ($p < .0005$). Specific comparisons of the three levels also show that the difference between PSP levels 0 and 1 are not statistically significant, while the difference between PSP levels 1 and 2 is significant ($p < .0005$).

7.3 Summary of Improvements in Yield

Early defect removal is one of the most economical ways to improve the quality of delivered software products. Preventing unplanned rework from occurring in the final stages of a software project allows more complete testing and assurance that the product will function as expected when it is delivered. These post-development effects are not directly observable in the PSP data (because the programming assignments are not delivered to a customer). However, the significant improvement in early defect removal is apparent.

The median improvement in yield was an increase of 50% in the number of defects removed before compile. This was computed by subtracting the yield for PSP level 0 from the yield for PSP level 2 for each engineer, then computing the median of that distribution. Half of the engineers improved their yield by more than 50%.

To illustrate improvement of yield in more concrete terms, data from a single engineer are presented below.

Assignment	Defects Injected Before Compile	Defects Removed Before Compile	Yield	Trend in Pre-Compile Defect Yield
1	5	3	60%	
2	7	0	0%	
3	7	0	0%	
4	2	0	0%	
5	3	0	0%	
6	1	0	0%	
7	3	3	100%	
8	3	2	67%	
9	4	3	75%	

PSP0 aggregate yield = 16%

PSP2 aggregate yield = 80%

Yield improvement = 64%

Table 7-1: Sample Data for Pre-Compile Defect Yield

8 Productivity

Productivity is a major focus of most organizations that produce goods for customers. The quantification of product output per unit of time spent is as old a metric as can be found in any industry. In PSP training, the data collected by the engineers allow them to compute lines of code per hour (LOC/Hr) as a measure of their personal productivity.

The hypothesis to be tested in this section is

> *As engineers progress through the PSP training, their productivity increases. That is, the number of lines of code designed, written, and tested, per hour spent increases between the first and last assignments.*

8.1 Group Trend

Figure 8-1 shows the trend in average productivity across the three PSP levels, with the dotted lines representing ± 1 standard deviation. Data from 196 engineers reporting actual time and actual product size are used in the figure and subsequent analysis in this section.

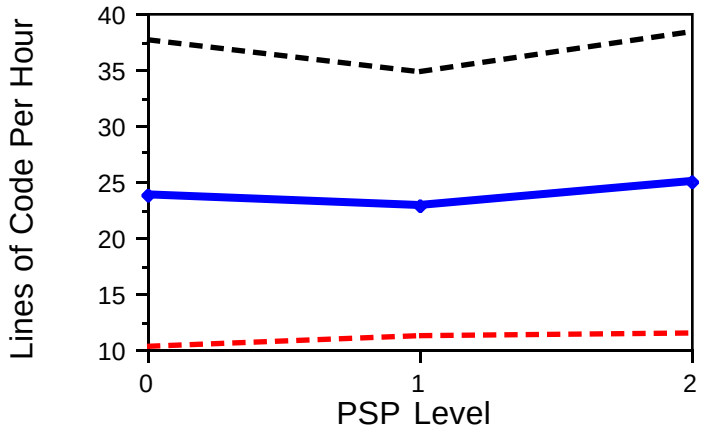


Figure 8-1: Average Productivity

The trend line in the figure above shows very little fluctuation in average productivity across the three PSP levels. The average hovers near 25 LOC/Hr for all three levels.

8.2 Analysis of Individual Changes in Productivity

The repeated measures ANOVA reveals that the differences in productivity across the three PSP levels is statistically significant (p = .012). Specific comparisons show that the difference between PSP levels 0 and 1 is not significantly different, but the difference between PSP levels 1 and 2 is significantly different (p = .004). However, when PSP level 0 is compared with level 2, the difference is not statistically significant. This set of statistical results, while interesting from a statistical point of view, is barely meaningful from a substantive point of view. To illustrate this point, Table 8-1 lists the average productivity for each level.

PSP Level	Average Productivity
0	23.927 LOC/Hr
1	22.958 LOC/Hr
2	24.899 LOC/Hr

Table 8-1: Average Productivity

Examining the averages, we see that there is a reduction in productivity of approximately one line of code per hour between levels 0 and 1 (which is not statistically significant). Between levels 1 and 2, there is an increase in productivity of approximately two lines of code per hour (which is statistically significant). Finally, from level 0 to level 2, there is an increase in productivity of approximately one line of code per hour (which is not statistically significant).

8.3 Summary of Changes in Productivity

While the statistical workings of the repeated measures ANOVA are sensitive to the changes in individual productivity values (rather than the changes in group averages illustrated above), examination of individual changes reveals a similar pattern. We therefore conclude that although significant fluctuation in productivity occurred (statistically), no real substantive gain or loss in productivity was observed. This finding leads us to reject the hypothesis that productivity increases during PSP training.

9 Conclusions

The objectives of this study were to examine the effect of the Personal Software Process on the performance of software engineers, and to consider whether the observed results could be generalized beyond the study participants. Because the PSP was developed to improve individual performance through the gradual introduction of new practices, the study took a similar approach, examining the change in individual performance as these practices were introduced.

Our analyses grouped individual data by PSP process level and then examined the change in individual performance that occurred from level to level. Using this approach we found that the improvements in four dimensions: size estimation accuracy, effort estimation accuracy, product quality, and process quality, were statistically significant. No statistically significant change in productivity was found, and so we can state that the improvements observed in the other performance dimensions were achieved without any loss of productivity.

In conclusion, the analyses reported here substantiate that trends in personal performance observed during PSP training are significant, and that the observed improvements represent real change in individual performance, not a change in the average performance of the group. Furthermore, we are confident that the observed improvements are due to the PSP and can be generalized.

References

[Boehm 81] Boehm, B.W. *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.

[Ferguson 97] Ferguson, Pat; Humphrey, Watts S.; Khajenoori, Soheil; Macke, Susan; & Matvya, Annette. "Introducing the Personal Software Process: Three Industry Case Studies*." IEEE Computer 30*, 5 (May 1997): 24-31.

[Greenhous 59] Greenhous, S.W. & Geisser, S. "On Methods in the Analysis of Profile Data." *Psychometrika 24*, 2 (June 1959): 95-112.

[Harwell 92] Harwell, Michael R.; Rubinstein, Elaine N.; Hayes, William S.; & Olds, Corley C. "Summarizing Monte Carlo Results in Methodological Research: The One- and Two-Factor Fixed Effects ANOVA Cases*." Journal of Educational Statistics 17*, 4 (Winter 1992): 315-339.

[Humphrey 95] Humphrey, Watts S. *A Discipline for Software Engineering*. Reading, Ma.: Addison-Wesley, 1995.

[Mauchly 40] Mauchly, J.W. "Significance Test for Sphericity of a Normal N-Variate Distribution." *Annals of Mathematical Statistics 11* (1940): 204-209.

[Page 63] Page, E.B. "Ordered Hypotheses for Multiple Treatments: A Significance Test for Linear Ranks." *Journal of the American Statistical Association 58*, 301 (March 1963): 216-230.

[Tabachnick 89] Tabachnick, B. G. & Fidell, L. S. *Using Multivariate Statistics.* New York: Harper Collins, 1989.

Appendix A Descriptive Data

The data analyzed in this report are derived from 23 offerings of the PSP training course. As described above, this course is aimed at software engineering practitioners and has been offered in academic and industrial settings. Table A-1 provides information about the size of the classes that provided data for this report, as well as the types of classes included.

Class Size Category	Number of Classes
4 to 10	6
11 to 15	11
16 to 21	6

Class Type	Number of Classes
Instructor Training	4
Industry Setting	8
Academic Setting	11

Table A-1: Class Sizes and Types

As shown in the table above, the typical class size for PSP training is relatively small. Approximately half of the classes were offered in an academic setting (i.e., offered to graduate and undergraduate students), and a little less than half of the classes were given to practicing engineers in commercial software development organizations.

A.1 Availability of Data

Many of the SEI-trained instructors in the field have made tailoring decisions to accommodate the needs of the population that they serve. Although tailored versions of the course retain the fundamentals of the PSP, they have led to differences in the amount of data available for the analyses in this report. In addition, the drop-out rate for the course is relatively high. Table A-2 provides a summary of the data available on total effort, size, and defect counts for the 10 programming assignments.

Assignment	1	2	3	4	5	6	7	8	9	10
Effort	277	276	271	262	248	239	226	219	209	152
Size	265	274	271	262	248	239	226	219	209	152
Defects	277	273	267	259	244	235	224	214	202	150

Table A-2: Number of Engineers Reporting Totals by Assignment Number

In all, the data set used in this report contains data from 298 engineers. However, as shown in Table A-2, there is a wide range of difference in the data available from individuals in this group. The most notable difference is the small number of engineers providing totals for assignment 10. This occurred mostly because there were several instructors who tailored the class to exclude assignment 10.

To conduct most of the analyses presented here, data at a much finer level of granularity is needed. The following table shows one example of the availability of this more detailed data, so that the reader is aware of the sample sizes used. The table below lists the number of engineers who provided phase-specific effort data.

Assignment	1	2	3	4	5	6	7	8	9	10
Planning	268	271	267	261	248	239	226	218	209	152
Design	266	265	264	251	243	233	224	213	208	152
Design Review	2	4	6	4	13	15	138	132	124	149
Code	277	276	271	262	248	239	225	219	209	152
Code Review	2	7	8	10	20	22	223	219	208	152
Compile	277	276	271	260	246	239	222	214	209	151
Test	276	276	269	262	247	239	226	219	209	152

Table A-3: Availability of Phase-Specific Effort by Assignment Number

Several patterns in the table above are worth mentioning. First, while design and code reviews were not explicitly introduced until program 7, some engineers carried out these reviews, based on their past experience with them. Second, there was at least one class in the group that chose not to introduce design review.

Finally, it is clear that the pattern of missing data is not identical across the various life-cycle phases. That is, there are instances of engineers reporting effort on some phases but not others. This pattern is not unlike what most researchers face when the data collection goals and research goals are not the same. The data collected during PSP training was designed to help engineers monitor and improve their processes, not to help in writing this report.

A.2 Typical Values and Variation

To give the reader a sense of the typical values of the measures being analyzed in this report, the following series of charts is provided. In these charts, only engineers who provided complete data (all 10 assignments completed) are included.

In Figure A-1, the average number of new and changed lines of code produced for each of the 10 assignments is plotted by the solid black line, with the dotted lines above and below showing +1 and -1 standard deviation, respectively.

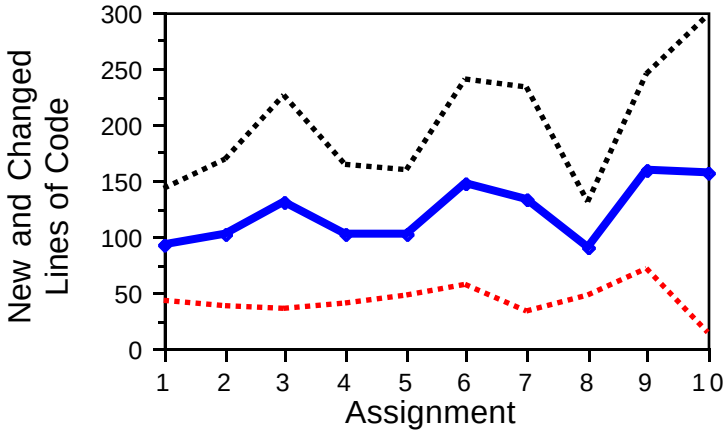


Figure A-1: Average Size

In Figure A-2, the average effort for each of the 10 assignments is plotted by the solid black line, with the dotted lines above and below showing +1 and -1 standard deviation, respectively.

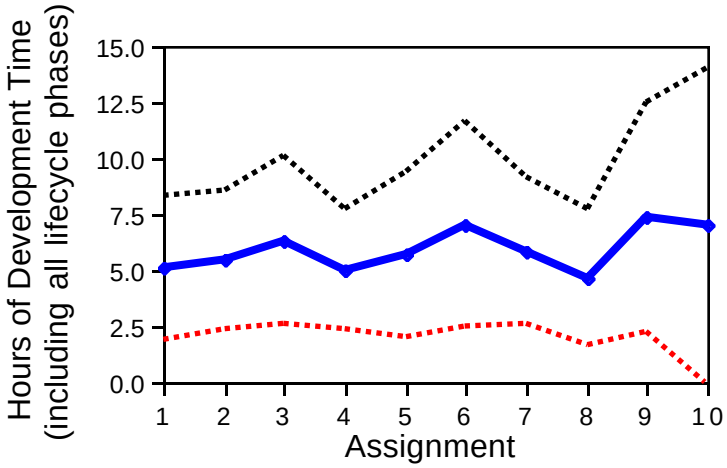


Figure A-2: Average Effort

As shown above for each of the three PSP levels studied, the third programming assignment (i.e., assignments 3, 6, and 9) is larger and takes more time than the previous two assignments.

The defect densities and productivity achieved by the engineers tend to vary from program to program as well. The average overall defect density is plotted by the bold line in Figure A-3, with the dotted lines above and below showing +1 and -1 standard deviation, respectively.

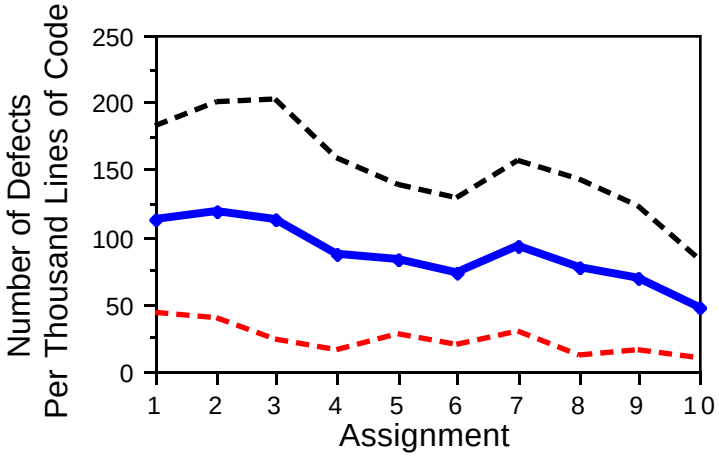


Figure A-3: Average Defect Density

The average productivity is plotted by the bold line in Figure A-4, with the dotted lines above and below showing +1 and -1 standard deviation, respectively.

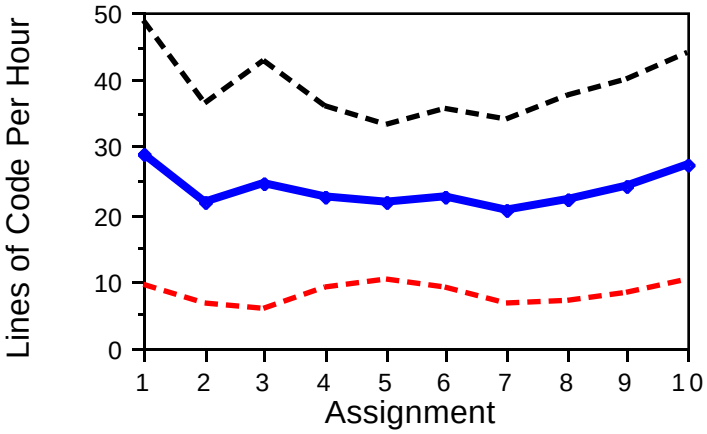


Figure A-4: Average Productivity

Figures A-3 and A-4 do not suggest any obvious similarities, as we saw for Figures A-1 and A-2. For overall defect density (Figure A-3), a decreasing trend across the 10 programming assignments is visible, with a notable increase for assignment 7. The trend for productivity (Figure A-4) suggests no marked increase or decrease overall. A slight dip in productivity during the middle of the assignment sequence is apparent, with the last assignment showing an average productivity very close to that of the first assignment.

A.3 Data Used in Specific Analyses

Because of the varying amounts of data that are available across the different measures used in PSP assignments, the sample size used in the analyses presented earlier in the report varies from assignment to assignment. The table below lists each analysis and the number of engineers on which they are based.

Report Section	Sample Size
6. Size Estimation	170
7. Effort Estimation	170
8. Defect Density	181
9. Pre-Compile Defect Yield	188
10. Productivity	196

Table A-4: Sample Size for Each Analysis

The limited sample sizes in each analysis (i.e., less than the total 298) derive from the fact that the focus of the analysis is on tracking the performance of individual engineers as they progress through the training. Therefore, if engineers do not report data for one or more assignments in the series, it is impossible to say what their progress was with the level of accuracy desired for this type of analysis.

The sample sizes shown in Table A-4 are the result of selection schemes to ensure appropriate use of data where incomplete records exist. These selection schemes, and the rationale that drove them, are described below.

A.3.1 Data Used in Analyses of Estimation Accuracy (Size and Effort)

First, because size estimates and effort estimates both play a central role in planning, and because effort estimates are derived through the use of size estimates, only cases where both types of estimates were available are used in the analyses.

Second, because the programs being estimated are generally small in size (see Figure A-1), the estimates and actuals are pooled by PSP level. That is, for each level, an estimation accuracy value is computed by summing the estimates across the three assignments, summing the actuals across the three assignments, and then computing [(Estimate - Actual) / Estimate]. Rather than using an average of three assignments within the level, this pooling serves to reduce the magnitude of outliers and provides a data point tied to performance within a PSP level. Thus, a “lucky guess” for the size of a given program (or effort required to produce it) will have less impact on the measure of performance for that PSP level.

Third, because of the focus on PSP level differences and the fact that PSP level 3 contains only one programming assignment, assignment 10 was excluded from all analyses of estimation accuracy.

Fourth, because size estimates are not required for the first programming assignment, their absence was not used to exclude cases.

As a result, a subset of 170 cases (from the total of 298) was used in the analyses of size and effort estimation accuracy. All of these cases provided estimated and actual effort for all nine assignments, actual size for all nine assignments, and estimated size for assignments 2 through 9.

A.3.2 Data Used in Analyses of Defect Density

For the analysis of defect density, the first nine programming assignments were used for the same reasons as stated above. In order to be included in the analysis, defect removal counts for the compile and test phases, as well as total defect counts, had to be available. In addition, because defect densities are calculated using the actual size of the programs, actual program size had to be available.

Instances where the total number of defects removed and the total number of defects injected are not equal presented a difficult problem in the analysis. Such differences could arise as a result of simple data entry errors, or they could be due to a data entry error related to reuse of code. For example, if code from assignment 5 is reused in assignments 6 and 7, engineers who uncovered defects in that code are supposed to record them in assignment 5 or in the assignment they are working on, depending on whether the code is categorized as “reused” code or “base” code.

Given that all data reported to the SEI derives from the spreadsheets completed by the instructors (who are working with the paper forms submitted by the engineers), we expect that some instructors were more careful than others in updating data from previous assignments, based on defect detection in later assignments. Furthermore, we expect that some percentage of the engineers in the PSP classes were inconsistent in the way they reported data in these situations.

While the lack of exact correspondence between the number of defects injected and the number removed presents an issue of precision, it seems reasonable to allow for some level of noise in the data being analyzed. For this reason, a threshold of two defects was established; that is, as long as the discrepancy between defects injected and removed was two or less, the data were included in the analysis. It seems reasonable to conclude that such small discrepancies arose from simple arithmetic errors or from loss of defect data when instructors failed to revise the spreadsheet upon discovery of defects during reuse of code.

Given these selection criteria, Table A-5 contains a listing, by assignment number, of the number of engineers who reported valid data. The first row lists the number of engineers providing sufficient data for each program. The second row lists the number of engineers reporting defect counts that differed by two or less (injected versus removed). The third row lists the number of engineers failing to report actual size. The fourth row lists the number of engineers whose defect counts (injected versus removed) differed by more than two defects. In all, 181 engineers provided complete data for all nine assignments by these criteria.

	Assignment Number								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Complete data	255	258	257	260	237	234	215	211	200
Defect counts off by 2 or less	8	6	12	1	10	4	9	6	8
No size reported	33	24	27	36	50	59	72	79	89
Defect counts off by more than 2	2		2	1	1	1	2	2	1

Table A-5: Availability of Data for Defect Density Analysis

As was the case for the analysis of estimation accuracy, the defect densities used in the analyses were computed by first pooling the defects and product sizes across the three assignments within each PSP level, then computing defect density as [total number of defects / (total new and changed LOC/1000)].

A.3.3 Data Used in Analyses of Pre-Compile Defect Yield

The analysis of pre-compile defect yield was conducted using the first nine assignments, for reasons discussed above. The yield value was computed based on pooled pre-compile defect injection and removal counts, as discussed above.

Because the same criteria were used (as above) for cases where the number injected differed from the number removed, the pooling of defect counts across assignments provided some protection from spurious values of yield (i.e., in a case where the number of defects removed exceeded the number injected, a yield value above 100% would be possible).

Once again, only cases with data from all nine assignments were included in the analysis. Because zero is a valid defect count in the spreadsheets submitted to the SEI, data were selected based on whether or not actual time was reported for each assignment.

There were 188 cases that met all these criteria, and the charts and analyses presented in Section 9 are based on this subset.

A.3.4 Data Used in Analyses of Productivity

The elements of the productivity metric consist of the number of new and changed lines of code, and the time spent completing the programming assignment. Therefore, only cases that contained both of these elements for all nine assignments were included in the analysis.

The pooling method described above was used to compute productivity values for each PSP level. Therefore, the value of productivity for PSP level 0 was calculated by summing the new and changed LOC of programs 1, 2, and 3, and then dividing that sum by the total time spent writing all three programs.

Appendix B Statistical Methods

This appendix assumes that the reader has some fundamental knowledge of statistics, such as might be gained from a series of undergraduate courses or a single graduate course focused on the general linear model (or sum of squares-based statistics). The main goal in providing this material is to allow other researchers in the field to understand fully the work presented here, in the hope that this understanding will serve to motivate further work in this exciting area of research.

For more information about the statistical methods used here, see the References on page 45 of this report for sources used in conducting this study. In particular, *Using Multivariate Statistics* by Tabachnick and Fidell [Tabachnick 89] is quite useful.

B.1 Repeated Measures Analysis of Variance

The primary statistical technique used to test the impact of PSP training in this report is the repeated measures ANalysis Of VAriance (ANOVA). This technique is well suited for a situation in which measurements are taken repeatedly on the same subjects. The following scenario describes one of the motivations for use of this statistical model.

> *In the study of the effect of a training course, a researcher may wish to compare scores on a pretest with scores on a test administered after training. Differences between these two test scores (given a host of other experimental conditions) are then attributed to the effectiveness of the training. In order to make a generalizable statement from such a study, individual differences must be accounted for before the two sets of measurements can be meaningfully compared. To simply state that the average post-test score for the group is greater than the average pretest score could overlook the possibility that the majority of subjects did not change at all, but one or two subjects scored significantly higher on the post-test than they did on the pretest.*

By performing the statistical test on the average change of the individuals (rather than the change in group average), the repeated measures ANOVA provides a more rigorous analysis of data collected over time from a single group of individuals. This additional level of precision, however, places more stringent requirements on the data collection and interpretation process. These requirements (referred to as assumptions in the statistical literature) are addressed below.

B.1.1 Assumptions Underlying the Correct Use of Repeated Measures ANOVA

In the broadest terms, the statistical model assumes that

- subjects are representative of the population of interest and are randomly selected
- observations on these subjects are independent (from subject to subject)
- dependent variables are normally distributed in the population
- the variance-covariance matrices of dependent variables are identical

A great deal of published work in methodological research suggests that statistical tests of this type are robust to violations of these assumptions [Harwell 92]. Unfortunately, many researchers cite those findings and proceed to completely ignore the opportunity to better understand their own statistical results. This section of the report addresses the condition of our data with respect to these issues and explains the ways we have attempted to overcome some of the issues deriving from violations of assumptions underlying the statistical model.

B.1.1.1 Representative Sample

Clearly, the sample used in this study is not a random sample from the population of all software engineers. In the most conservative application of statistics, the generalizability of our findings is limited to the group of engineers who provided the data analyzed here. The sampling used here is often termed a “convenience sample,” and most experimental design texts strongly caution readers to resist overgeneralization. Unfortunately, most authors offer very little advice beyond these cautions.

The reasonable boundaries of generalization fall somewhere between the two different populations (*all* engineers, versus the 298 providing data). Our claim is that future offerings of PSP training will yield results of the type and magnitude reported here. Surely we would be uncomfortable with generalizations to PSP-like courses that omitted key lessons, or that involved radically different exercises. While we have not explicitly identified the vital ingredients that make PSP courses and their outcomes similar, the experience to date confirms that the core “technology” being transferred to SEI-trained instructors is repeatable. We base our claim of generalizability on the similarity of class trends presented by each instructor, as well as the results presented here. Therefore, while the sample used in the analyses described in this report is not a random sample of software engineers, we feel confident in claiming that engineers trained in PSP (using the SEI training course) will achieve improvements similar to those documented here.

B.1.1.2 Independent Observations

In order for this assumption to be satisfied, we need to establish that the data collected from one engineer is not (in some sense) dependent on the data from another engineer. A classic issue which arises in this area is the effect of different instructors. In many settings, the performance of students in a class is limited or enhanced by the quality of the instruction they receive. Other examples of issues associated with independence of observations include

gender, educational level, professional experience, and programming language used. When variables outside the statistical equation can “account for” observed differences (or similarities) in the data collected, the possibility of non-independent observations should be explored.

In the most conservative application of statistics, such independence is rarely (if ever) achieved. We do not claim that all potential confounding effects have been examined here. We have taken care to get a large sample of engineering data, generated by engineers with a variety of instructors, in hopes that such effects will be minimized.

B.1.1.3 Normally Distributed Populations

The assumption of normality is perhaps the most frequently violated assumption in the application of ANOVA. Fortunately, this is typically the least costly error in cases where large samples of data are analyzed. The Central Limit Theorem protects the researcher from being terribly misled by the statistical results from moderately non-normal data.

In our analyses, we applied a multivariate (as opposed to a univariate) model to the data. This means that the population distributions are assumed to be multivariate normal distributions. Examination of plots and simple statistics (like skewness and kurtosis) are not generally useful in checking for multivariate normality. Furthermore, examination of each variable for bivariate normality actually does not provide protection against the violation of the multivariate normality assumption.

Our approach in assessing the tenability of the multivariate normality assumption is to examine the distribution of residuals in each analysis. A normal probability plot is drawn for each dependent variable, and obvious deviations from normality are sought.

As shown in most of the figures in this report, much of the data we are working with tends to follow a skewed distribution. In fact, our examination of normal probability plots for residuals confirms that the normality assumption is violated for many of the analyses. In these instances, we performed transformations on the original data and replicated the analyses with the transformed scores. When the statistical results derived from the transformed variables are consistent with those of the raw variables (and the plot of residuals indicates a more nearly normal distribution), we can be confident that the lack of normality is not leading us astray. These transformations and confirmatory analyses are described more fully in Section B.3.

B.1.1.4 Identical Variance-Covariance Matrices

Because we are using a multivariate treatment of the data in performing our omnibus tests, the issue of variance-covariance homogeneity becomes moot. The measures from each engineer are treated as an observed vector from a multivariate distribution, rather than a set of observed data points from a family of univariate distributions.

B.2 Post-Hoc Analyses

In each analysis, significant multivariate tests are followed by examination of univariate ANOVA tables comparing pairs of PSP levels. In shifting to a reliance on the univariate approach an additional assumption is placed on the condition of the data. The population variance-covariance matrices are assumed to be homogeneous. While the multivariate approach used in the omnibus test relaxes this assumption, proper interpretation of the univariate results is dependent on this assumption.

In order to ensure that violations of the homogeneity of variance-covariance matrices assumption were not causing us to misinterpret our results, we tested the sample data and took appropriate steps to counteract the impact. We tested the appropriateness of this assumption using Mauchly's test of sphericity [Mauchly 40]. In cases where heterogeneity of population variance-covariance matrices is suspected, several corrections are available to combat the potential for misleading significance levels. The most conservative of these methods is the Greenhous-Geisser epsilon [Greenhous 59]. Significant pairwise comparisons reported in this paper rely on Greenhous-Geisser corrected F-tests, if the test for sphericity was rejected.

B.3 Confirmatory Analyses Using Transformed Data

As discussed in Section B.1.1.3, the sample distributions of dependent variables analyzed in this study are typically non-normal in shape. While this fact alone does not necessarily mean the multivariate normality assumption is violated, examination of normal probability plots of the residuals did suggest this as an issue for every analysis conducted.

In order to ensure that violations of the normality assumption are not causing gross misinterpretations, we repeated most of the analyses using transformations of the dependent variables that resulted in more nearly normal distributions. This allowed us to confirm that significant differences observed based on "raw scores" do, in fact, reflect reliable patterns in the data. Another option would be to work with the transformed data alone. However, when discussing variables like the ones used here, the interpretation of the transformed data may be questionable. For example, the log-transformed value of defect density means nothing to the engineer who collected the data. In order for it to be usable in monitoring the engineer's personal process, the metric must be expressed in its "raw" form. Given this fact, significant differences using the "raw" metrics have meaning, and significant differences using transformed metrics lend support to this meaning (while ensuring that the necessary preconditions for use of the statistical procedures have not been abused).

The two sets of analyses ("raw" and "transformed") were compared to detect three critical differences. First, the normal probability plots were compared to confirm that the transformation did in fact result in a more nearly normal distribution of residuals. Second, the alpha levels were compared to ensure that the type-I error rate was below .05 for both analyses. Finally, the power values (computed for a nominal alpha level of .05) were compared to ensure that the power of the F-test was not grossly deflated (i.e., below .5).

Each of the transformation techniques used is discussed in detail below.

B.3.1 Transformations Used to Confirm Analyses of Estimation Accuracy

The careful reader will note that the definition of estimation accuracy (for both size and effort) yields values that are not symmetrical around zero. The value of estimation accuracy is calculated by dividing the difference between actual and estimate by the estimate. As a result, there can never be an overestimate of greater than 100%.[1] This definition of the dependent variable will result in a negatively skewed distribution almost by definition. In essence, for a given number of lines of code (or minutes of development time) misestimated, the distance of the estimation accuracy value from zero will differ depending on its sign. Furthermore, this difference is particularly exaggerated when the estimate is very small.

In order to combat the undesirable effects[2] of this phenomenon on the statistical procedure, the confirmatory analysis was conducted using a new definition of estimation accuracy. For underestimates, the metric was computed using the actual in the denominator, while for overestimates, the metric was computed using the estimate in the denominator. The result is a more nearly normal distribution of "estimation accuracy" with a possible range of -1 to +1. Examination of the normal probability plots confirmed that the transformation resulted in a more nearly normal distribution of residuals as well.

B.3.2 Transformations Used to Confirm Analyses of Defect Density

As Figure 6-2 on page 34 reveals, the distributions of defect density tend to be positively skewed, with long tails extending to the right and a truncated range at zero. This type of non-normal distribution is to be expected given the source of the data. There can never be a negative count for defects, so the truncation at zero is expected. In addition, we would expect many small values of defect density and relatively fewer large values. This positively skewed distribution is particularly expected when engineers (as a group) reduce the defect density of the programs as they improve their quality during the course.

This is the type of data where either a logarithmic or inverse transformation can be used to create a more nearly normal distribution [Tabachnick 89, page 85]. Based on our examination of the effects of these two types of transformations on the distribution of residuals, the logarithmic transformation was used in the confirmatory analysis.

---

1. Consider an extreme example, where the actual is infinity and the estimate is one. The value of estimation accuracy approaches negative infinity (negative infinity divided by 1). On the other extreme, if the actual is 1 and the estimate is infinity, the value of estimation accuracy approaches 1 (infinity divided by infinity).

2. This is not an undesirable effect for the operational use of the metric itself, as it is intended to quantify the 'amount of error' in the estimate and not its distance from the actual.

B.3.3 Confirmatory Analysis of Yield

The distributional forms of the three yield values (one pooled value for each PSP level) presented the most challenging condition of non-normality. As Figure 7-2 on page 38 illustrates, the yield values for PSP levels 0 and 1 were consistently low, and only during PSP level 2 did they begin to more fully reflect the entire range of possible values. This presents us with two highly skewed distributions (PSP levels 0 and 1) and one fairly symmetrical distribution (PSP level 2). We are hard-pressed to devise a single transformation that can be applied to all three of these distributions in order to make them all more normal simultaneously. Examination of the normal probability plots with the residuals from the ANOVA confirmed that the distribution for PSP level 2 is fairly normal in form, whereas the distributions for PSP levels 0 and 1 are grossly non-normal.

Given this condition, we rely on a nonparametric alternative to the repeated measures ANOVA to perform the confirmatory analysis. In general, where the assumptions of the parametric test can be met, procedures like ANOVA tend to be more powerful than their nonparametric counterparts. However when the assumptions are not tenable, the nonparametric alternative can often be the most powerful technique.

The Page test for ordered alternatives [Page 63] provides an alternative to the parametric ANOVA technique and also allows a directional alternative hypothesis. In other words, this technique allows us to test for significant differences across PSP levels without the assumption of multivariate normality, and it allows us to test whether or not yields in successive PSP levels are higher than previous levels.

Results from the Page test indicate statistically significant ordered differences in yield from PSP level to PSP level ($p < .00003$).[3] This analysis confirms the statistically significant increase in yield documented in Section 7.

B.3.4 Transformations Used to Confirm Analyses of Productivity

The distributions of productivity for each PSP level show moderate positive skew. Again, this type of distribution is to be expected, given the source of the data. There can never be a productivity value of zero, since that would translate to zero lines of code per hour. While vast differences in individual productivity exist among engineers, the number of very high productivity values expected (versus very low) is relatively small. Therefore, we see many engineers reporting moderate to low values of productivity, and a few engineers reporting relatively high values of productivity.

3. The actual p-value is likely to be much smaller. This test was conducted ‘by hand,’ as our statistical package does not provide for it. The smallest value in the table we used had a p of .00003 for a test statistic of 4; our test statistic was 12.6!

This type of “non-normality” is often treated with a square-root transformation. The normal probability plots of residuals for the original analysis (using “raw” values of productivity) indicated a departure from normality, so the confirmatory analysis was carried out with square-root-transformed productivity values. Examination of the normal probability plots (from the analysis of transformed data) confirmed that the transformation did, in fact, result in more nearly normal distributions of residuals.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. **agency use only** (leave blank)	2. **report date** December 1997	3. **report type and dates covered** Final
4. **title and subtitle** The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers		5. **funding numbers** C — F19628-95-C-0003
6. **author(s)** Will Hayes, James W. Over		
7. **performing organization name(s) and address(es)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. **performing organization report number** CMU/SEI-97-TR-001
9. **sponsoring/monitoring agency name(s) and address(es)** HQ ESC/AXS 5 Eglin Street Hanscom AFB, MA 01731-2116		10. **sponsoring/monitoring agency report number** ESC-TR-97-001
11. **supplementary notes**		
12.a **distribution/availability statement** Unclassified/Unlimited, DTIC, NTIS		12.b **distribution code**

13. **abstract** (maximum 200 words)

This report documents the results of a study that is important to everyone who manages or develops software. The study examines the impact of the Personal Software Process (PSP) on the performance of 298 software engineers. The report describes the effect of PSP on key performance dimensions of these engineers, including their ability to estimate and plan their work, the quality of the software they produced, the quality of their work process, and their productivity. The report also discusses how improvements in personal capability also improve organizational performance in several areas: cost and schedule management, delivered product quality, and product cycle time.

14. **subject terms** capability maturity model, empirical studies, personal software process, process improvement, PSP			15. **number of pages** 62 pp. 16. **Price Code**
17. **security classification of report** UNCLASSIFIED	18. **security classification of this page** UNCLASSIFIED	19. **security classification of abstract** UNCLASSIFIED	20. **limitation of abstract** UL