

An Experience Report on the Personal Software Process

Individual developers can use quality analysis and management techniques that many consider applicable only to projects and organizations. One software practitioner explains how the Personal Software Process gave him the training he needed.

Jagadish Kamatar, *Advanced Information Services*

Will Hayes, *Software Engineering Institute*

I began my Personal Software Process training in July 1995 but didn't complete the two-week course until May 1996. After the first week, I was assigned to a project; four weeks later, when the second week of PSP training was scheduled, my project responsibilities took priority. So, I decided to wait until the next offering of the course to complete the training.

Using what I knew of the PSP on the first project was an interesting experience. I completed enough of the course to appreciate the data collection regimen and saw some of the benefits of enhanced predictability. (The sidebar on PSP training provides more background). Figure 1 shows the sequence of events from the start of training through the first two projects on which I used the PSP. Each diamond in the figure represents a milestone where a code module was complete and ready for integration (the figure does not show other project milestones). In all, I built 26 components (across two projects) during a three-year period.

Looking back, I would have benefited from the second week of training before getting so far into project work. I completed

the second week before my first project was finished, but changing my approach in mid-stream did not seem feasible. Therefore, I did not apply the new ideas I learned from the latter half of the course until the second project came along.

My performance on the class assignments was typical of the experience reported by the Software Engineering Institute.¹ My estimation accuracy improved, my ability to eliminate defects early improved, the defect density (defects per 1,000 lines of code) of the programs I wrote declined, and my productivity fluctuated between 17 and 32 lines of code per hour during the training. On the final programming assignment, I underestimated the size by only 4% and overestimated the effort by only 6%. On the last

PSP Training

The beneficial impacts of PSP training on individual software engineers are documented in an SEI technical report.¹ The experiences described in this article go beyond the classroom into the real-world application of these techniques. One of the authors, Jagadish Kamatar, is a practicing software engineer who has been using the PSP for several years. Jagadish contributed data to the original study focused on training impacts. The other author, Will Hayes, is a software engineering researcher at the Software Engineering Institute. Will is one of the authors of the first PSP impact study.

This article describes Jagadish's experiences applying PSP principles in two industry projects. Jagadish asked Will to turn his experiences into this report.

two assignments, I had error-free compiles on the first try.

PSP on the Job: The First Project

A large portion of my company's work takes the form of contracts, where our engineers work within a project managed and located at the customer site. Not coincidentally, my ability to accurately predict a completion date for the work I am assigned is a very important measure of my performance. I was among the first few groups of engineers to be trained in the PSP in my organization. Efforts to improve the software development processes in my organization had been underway for several years. The PSP seemed a natural fit for us, as an extension to the organizational improvement efforts that were underway.

Four engineers were to implement a product that had already been through the requirements and high-level design phases. The project manager was fully PSP trained, I had finished half of the training, and another project member began the PSP training during the project. In addition to our

personal process steps, we drew from the development methods defined for use in our organization. Detailed project management techniques, as well as engineering standards, were made to work with the new PSP techniques. My colleagues and I performed inspections on the requirements document, the high-level design specification, the detailed designs, and the code modules.

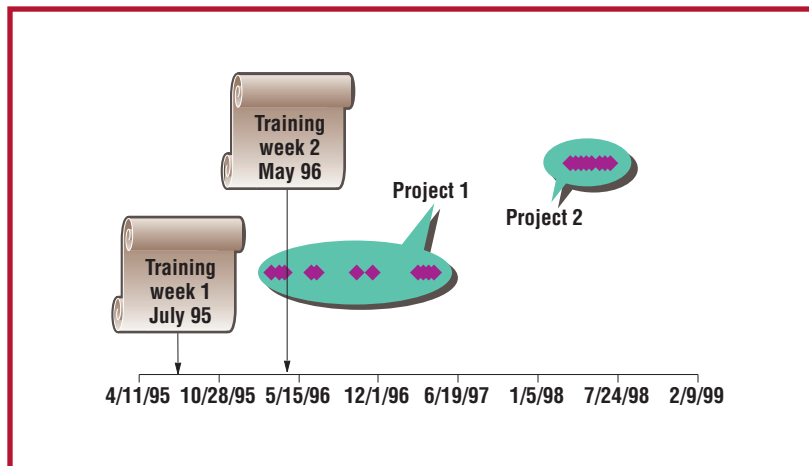
My initial use of PSP methods was rather limited. I focused on collecting accurate data and making good estimates. Predictability was perhaps the most important performance area for me professionally. Nevertheless, because I began using the PSP prior to receiving the lessons on personal quality and defect analysis, I completely overlooked the quality indicators and how they related to predictability. Even after completing the rest of the course, I found it difficult to analyze my quality data.

Despite this fact, when the customer performed acceptance testing on the product we built, they discovered only three defects in approximately 40,000 lines of code. I felt that I performed well on my first application of PSP. I was able to collect accurate data, though I might have done more to use it. Data collection was difficult at times; I had to record data in a variety of media (paper notes, spreadsheets, and my notebook) and then transfer it to a spreadsheet during the postmortem phase for each component. I also struggled with consistent defect classifications. My company's inspection training included an overview of the company standard for defect classifications. While this standard did not necessarily conflict with the classification scheme used in the PSP,² the ambiguities in each scheme sometimes led to confusion.

I overestimated project effort by about 10%. The accuracy of my estimates for individual components varied; I overestimated some and underestimated others. However, the Proxy-based estimation method² let me construct estimates whose errors tended to cancel each other out, rather than compounding the problem across the components I built.

One standard for judging estimation accuracy is based on the percentage of modules for which the effort was estimated within roughly 35%. For this first project, 68% of the modules had effort estimates that met

Figure 1. PSP adoption timeline. Each diamond represents a milestone where a code module was complete and ready for integration.



this criterion. The threshold for good estimating performance is sometimes set at 75% of modules being estimated within roughly 35%.

What I Overlooked the First Time

The lessons of PSP training were compelling and understandable, but I had trouble making the connection between these ideas and the real project setting. The quality management techniques were perhaps the most important missed opportunity. My experience with inspections on projects was sufficient to demonstrate their value; we even train our customers in the inspection method we use. I had received this training and had some experience inspecting specifications and code on a previous project. Tailoring the process to use historical data to guide my personal inspections seemed like too much effort for minimal return—given my successful experience with the existing inspection method. My focus on the current project's schedule kept me from taking on this extra task.

The postmortem phase, as described in PSP training, is devoted to examining data and considering improvement opportunities. During the training, I was so anxious to get everything finished and move on to the next assignment that I typically rushed through this final phase of the life cycle. Given the daily pressures of the real work environment, I found it even more difficult to take full advantage of this phase. I neglected the analysis of the data I collected except in the area of predictability. It was important to ensure that I knew the cause of inaccurate estimates so I could watch for those issues in the future.

Looking back, I realize that in adapting the new personal processes to the project setting in my organization, I was successful. However, I had not settled on an approach toward analyzing the defects in the products I built.

Defect Analysis

Shortly after completing the training course, I was assigned a process coach—a project manager well versed in the PSP. After reviewing my data, my coach pointed out that although I seemed to spend an appropriate amount of time in design reviews and code reviews, the number of defects re-

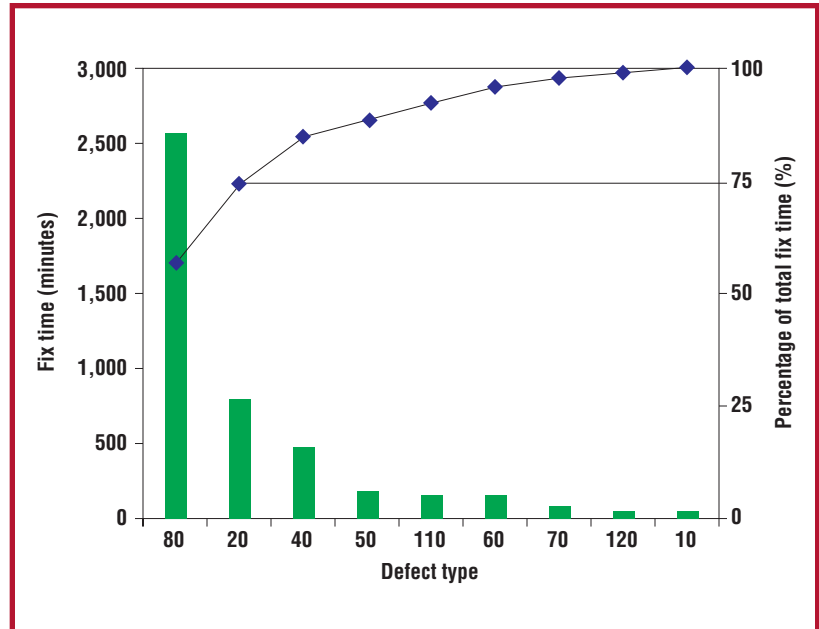


Figure 2. Pareto analysis defect fix times by defect type. The horizontal line indicates that the first two defect types accounted for approximately 75% of total rework effort.

moved in these reviews seemed low. When I described the review process I followed, my coach suggested I examine the data and create review checklists tailored to my defect history. By looking for the mistakes that occur frequently and consistently create rework, I would be able to save time and ensure higher quality. Isolating the largest source of rework in my defect history was not very difficult.

Using my electronic defect log (a simple spreadsheet template), I tabulated the fix time associated with each type of defect. As a result, the Pareto analysis of defect types with their associated total fix times, shown in Figure 2, was my first defect analysis on the job.

As the figure shows, type-80 defects accounted for the largest amount of rework effort at the time of analysis.² Type-80 defects are function defects related to an error in logic, use of pointers, loops, recursion, or computations. Together with type-20 defects (syntax errors involving spelling, punctuation, instruction formats, and typing mistakes), these defects accounted for about 75% of my rework effort on the first project. Type-20 errors occurred more frequently (99 times), but the 89 type-80 defects represented more rework effort and would likely cause more problems if they escaped to the field.

Armed with this information, my process coach and I read the descriptions I had

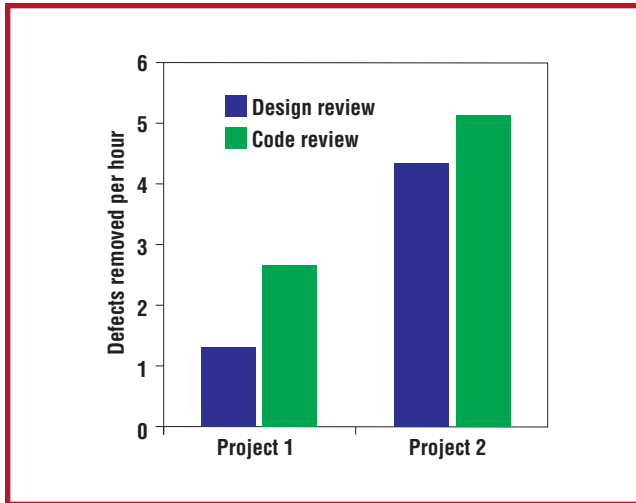


Figure 3. Defect removal rates.

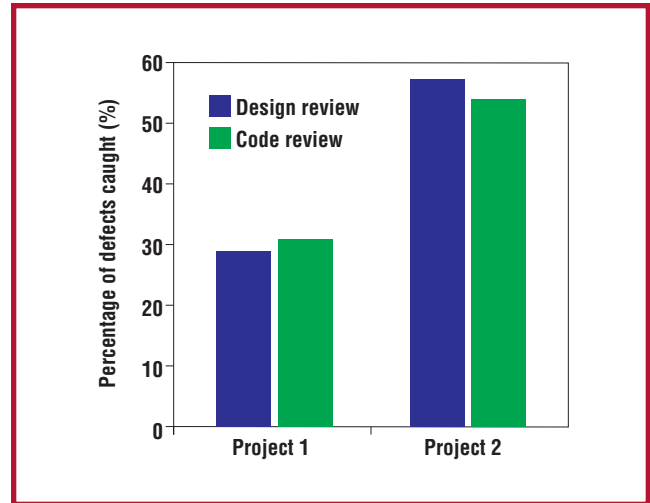


Figure 4. Review yields.

written in my defect log for type-20 and type-80 defects. We constructed ways to look for these defects during design and code review, and before I finished reading all of the defect descriptions, I had created two review checklists, one for design and one for code, to use in developing the next component. My goal was to make better use of the time I spent doing design and code reviews and to develop higher-quality products from the earliest phases of the personal life cycle.

Industry standards on the effectiveness of personal reviews (like the ones used in the PSP) do not exist. However, research on the effectiveness of reviews and inspections at the project and organizational level provide some targets to consider. Barry Boehm reviews this data in his discussion of the Software Error Introduction and Removal Model.³ The percentages reported for design-related inspections and reviews range from 40% to 70%, with most study results showing 50% or less. The performance of code inspections, on the other hand, tends to be consistently above 60%, with some exceptions.

Experimental results on the effectiveness of inspection methods often focus on new reading techniques to increase the percentage of defects detected.⁴ In these experimental studies, it is not uncommon to observe defect detection rates well below 50%. A new method that can detect 60% of the defects can prove to be significantly better than the baseline.⁵

Using My Improved Personal Process: The Second Project

My second project using the PSP differed from the first, in that I was now a member of a customer-managed project. Without the regimen of inspections and the project processes available to me in my home organization, I was much more focused on the quality of the components I built. My newly created checklists were important tools for me in this context, and after building each component, I reviewed my defect log with my checklist in hand. I wanted to ensure that the defects I intended to catch using the checklists were indeed being caught.

The amount of time I spent in the post-mortem phase increased for this second project. In the first project, I spent an average of 6% of my total development time in postmortem, entering my data into spreadsheet templates and looking for anomalies. On the second project, I spent an average of 9% of my total development time in post-mortem. With explicit focus on quality added to my process for this phase, I was getting more out of the time I spent entering and analyzing my data. Spending 1.5 to 2.5 hours in this final phase, building no product, could be difficult to justify. However, analysis of the data I collected quickly showed that this was time well spent.

My personal reviews had improved in both effectiveness and efficiency. As Figure 3 shows, I removed more defects per hour during the reviews.

Looking at the data from the two proj-

ects, now that both are complete, I also see that my review processes caught a larger percentage of defects on the second project (as Figure 4 shows).

The effectiveness of my reviews seemed more important in this second project, as this team did not have the benefit of the type of inspection process that my organization uses. In addition, the integration and system testing process would have benefited from a more structured approach. The type of structure found in the Team Software Process, with well-defined roles and responsibilities, would have helped this project a great deal.⁶

My effort estimation accuracy for the second project was an aggregate 6.7% overestimate (compared to the aggregate 9.7% underestimate in the first project). Finally, I estimated 86% of the modules within roughly 35%. This reflects a noticeable improvement from the first project. The additional historical data in my database, as well as improvements in product quality, enabled more accurate estimation in the second project.

The total life-cycle yield (the percentage of defects removed before compile) was consistently above 50% (with a maximum of 93% for one component). The customer found only one minor defect during acceptance testing in the 2,671 lines of code I developed for the second project. I introduced the defect during my planning for one of the components, when I overlooked a data-formatting requirement. The customer had a work-around that resulted in the end user typing one additional number in a field on a data entry screen. Finally, with these improvements in predictability and quality, I enjoyed a 62% increase in productivity on the second project.

The software industry's demand to achieve predictability and consistency in the face of rapid change is significant. The PSP framework helps an individual to meet these demands.

Using the PSP has provided me with several benefits. My estimation accuracy has improved significantly. However, adding more data to my historical database will

help me further improve my estimating skills. As it is said, "there is no substitute for hard work to be successful." Similarly, "there is no substitute for more data to improve an individual's personal processes."

My current goal is to narrow the percentage error in my estimates to within roughly 5%. I also plan to focus on improving my early defect removal through more effective reviews and preventing defects by improving my skills and practices. ☺

References

1. W. Hayes and J. Over, *The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers*, Tech. Report SEI/CMU-97-TR-001, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1997.
2. W. Humphrey, *A Discipline for Software Engineering*, Addison Wesley Longman, Reading, Mass, 1995.
3. B.W. Boehm, *Software Engineering Economics*, Prentice Hall, Upper Saddle River, N.J., 1981.
4. V. Basili et al., "Empirical Investigation of Perspective-Based Reading," *J. Empirical Software Eng.*, Vol.2, No.1, 1996, pp. 133-164.
5. W. Hayes, "Research Synthesis in Software Engineering: A Case for Meta-Analysis," *Proc. Sixth Int'l Software Metrics Symp.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1999, pp. 143-151.
6. D. Webb and W.S. Humphrey, "Using the TSP on the TaskView Project," *CrossTalk: J. Defense Software Eng.*, Vol. 12, No. 2, Feb. 1999, pp. 3-10.

About the Authors



Jagadish Kamatar is a project manager at Advanced Information Services Inc., Peoria, Illinois. He is also an SEI-authorized PSP instructor and has trained several software engineers in the PSP at AIS and elsewhere. His research interests include software engineering, managing software projects, and software quality using metrics.

Kamatar received a bachelor's in engineering degree in electronics and communication from Karnatak University, India, and a master of technology degree in computer science and engineering from the Indian Institute of Technology, Bombay. He is a life member of the Indian Society for Technical Education and an associate member of the IEEE. Contact him at Advanced Information Services, Ste. 114, 1605 Candletree Dr., Peoria, IL 61614; jagadishk@advinfo.net.



Will Hayes is a senior member of the technical staff at the Software Engineering Institute. He is an SEI-authorized lead assessor and CMM instructor, as well as instructor of the SEI courses Goal-Driven Software Measurement and Statistical Process Control for Software. His research interests include empirical studies of the impact of software engineering practices, the validity and reliability of software process appraisals, and effective use of measurement in software development organizations.

Hayes received a BA in psychology from Washington College and an MA in research methodology from the University of Pittsburgh. Contact him at the Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890; wh@sei.cmu.edu.

Only one minor defect was found during customer acceptance testing in 2,671 lines of code.