The Personal Software Process
Overview, Practice, and Results[1]

by
Watts S. Humphrey
The Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

You would probably agree if I told you that the number of defects to be found in testing a program would be proportional to the number in the product when it entered test. It seems reasonable to find more defects when there are more to be found. If, however, I said that the number of defects in the product after test would be proportional to the number on test entry, many software engineers would not agree.

Software people act as if testing will find all or most of a product's defects. There is, however, compelling evidence that even well-run unit tests are less than 70 percent effective at finding defects. Integration and system tests only find about 45 percent and function tests typically find a dismal 8 percent of the product's defects. Thus, to get a quality product out of test you must put one in. If software engineers really believed this, they would act more like quality engineers in other fields. That is, they would concentrate on finding or preventing the defects before the start of test.

Many software engineers will argue that this approach is impractical for software development. There is, however, evidence that this strategy works. By using a defined and measured personal software process, engineers can improve the quality of their products by five to ten times while also improving their productivity. This personal software process (PSP) is a promising way for engineers to understand their own performance and to see how to improve it. The PSP is new, however, and there is limited experience with its introduction and use. It has been taught in six universities and experimentally applied by three software organizations.

The original impetus for developing the PSP came from questions about the Software Engineering Institute's (SEI) capability maturity model (CMM). Many viewed the CMM as designed for large organizations and did not see how it could be applied to individual

[1]This work is supported by the U.S. Department of Defense.

work or to small project teams. While the CMM does apply to both large and small organizations, more explicit guidance was clearly needed. The SEI thus started a process research project to examine ways individual engineers could apply level 5 process principles. After several years of research, means were devised to adapt 12 of the 18 CMM key process areas to the work of individual software engineers.

Experimental work was then started with several corporations to see how experienced engineers would react to the PSP and to explore introduction methods. It was found that experienced engineers are generally attracted by the PSP strategy and find the methods help them in their work. In the words of one engineer, "This isn't for the company, it's for me."

The PSP applies process principles to the work of software engineers by

- providing a defined personal process framework,

- introducing a family of process measures,

- using these measures to track and evaluate performance,

- striving to meet quality criteria and improvement goals.

In using the PSP, engineers

- develop a plan for every project,

- record their development time,

- track their defects,

- retain the data in project summary reports,

- use the data to plan future projects,

- analyze the data to evolve their processes and improve their performance.

In the PSP, project plans include a documented size estimate and a statistically derived size prediction interval. Historical data are used to estimate the development time and to calculate the time prediction interval. Based on the engineer's personal data on prior projects, these time data are spread over the project phases. Together with data on the engineer's prior commitments and available working time, the engineer then produces a schedule.

The development plan also includes a defect estimate. From data on their prior experience, the engineers learn to accurately project the defects they will inject and remove per phase. They also know the likely distribution of defect types and the likely times required to find and fix defects in review, compile, and test. As they track these data, the engineers develop review checklists to help find the defects earlier in their processes. They also look for ways to improve their processes so they can prevent the defects before they are introduced.

The quality strategy used with the PSP is consistent with that practiced in many hardware organizations: build quality into the product from the start. The general practice in software has been to design and implement products as rapidly as possible and then to rely on compile and test to find the defects. When organizations work this way they spend as much as half their development resources compiling and testing. Even after all this expense, quality is generally still so poor that extensive field tests are needed before products can be offered for general use. PSP engineers follow a different strategy. They have found that testing is inefficient and marginally effective. Their objective is to remove all defects before the first compile or test. One of the principal PSP quality measures is yield: the percent of all defects removed before the first compile or test.

The results to date from the available data on four PSP courses show that improvement is substantial and almost universal. For example, the percentage improvement in the average numbers of defects found per thousand lines of code (KLOC) from the beginning to the end of the 15-week PSP course shows improvements of from two to five or more times. The following table shows the percentage improvement in the total defects found in compile and test from the first to the last PSP exercise. These data are for all the students in four university courses.

Where found	**Class A**	**Class B**	**Class C**[2]	**Class D**

[2]Note that class C only did 9 of the exercises so the second set is the average of exercises 8 and 9.

Total defects	53.4%	45.8%	55.1%	80.1%
Compile defects	68.8%	76.6%	75.7%	88.1%
Test defects	68.8%	81.7%	64.2%	83.2%

In calculating these data, the average defects per KLOC for the first two programs was compared to the average for the last two. The numbers of engineers in classes A, B, C, and D were 4, 12, 6, and 19 respectively. The students in classes A, B, and C were moderately experienced engineers while most members of class D were graduate students with little industrial experience.

With these dramatic quality improvements, one might think that productivity would suffer. It was found, however that the students soon became quite efficient at gathering data and making plans. Even including all this overhead time, the engineers' productivity typically improved by 20% or more. It should also be noted that the last exercises were substantially larger and more difficult than the first. As the following table shows, the productivity increases varied considerable and were generally quite large.

Average LOC/Hour	**Class A**	**Class B**	**Class C**	**Class D**
Exercises 1 and 2	19.9	31.4	11.4	13.8
Exercises 9 and 10	36.3	38.6	26.9	22.3
Percent Improvement	82.4%	22.9%	136.0%	61.6%

While all groups improved, the amount of increase was more or less inversely proportional to the initial productivity level. This implies that there is some limiting rate for lines of code (LOC) per hour. This is analogous to the 4:00 minute mile where the difference between world record holders and competent runners is only a few seconds. Group productivity rates appear to converge on about 30 to 40 LOC per hour but I have seen individual rates as high as 85 LOC per hour. For 100 percent yield developments, however, the highest rate observed is 65 LOC per hour. While many factors will influence these rates and while some engineers will likely have higher rates for smaller or lower yield developments, the highest consistent rate for sustained high-quality work appears to fall somewhere around 70 LOC per hour. Substantially more data will be required, however, before such limits can be determined with any confidence.

Assuming that there is such a rate limit, engineers could then make significant initial productivity improvements by defining and tracking their personal processes. Thereafter, productivity increases would generally not come from producing more LOC per hour. While better languages might provide some benefits, we will likely need improved architectural design concepts and more effective ways to reuse standard program elements. There are also significant improvement opportunities in the requirements and system design phases and in finding better ways to integrate small programs into larger systems. The PSP has not yet been used in these areas but its principles are applicable.

One might ask why the software community has been so slow to adopt proven quality principles. The answer appears to be that these methods are difficult to introduce and are not intuitively obvious. Without convincing evidence, for example, few engineers believe it is more efficient to find defects by reviewing code than by testing and debugging. The PSP thus uses a phased introduction strategy to demonstrate the methods to the engineers with their own data. By following a seven step process, completing 10 small programming exercises, and producing five analysis exercises, engineers see how the PSP methods work for them.

Six university courses have been taught to test the PSP course in software engineering curricula. The results were so positive that two universities have made the PSP a required course in software engineering. One has even made it the first course students must take in their software engineering masters degree program.

In developing the PSP industrial introduction strategy, we have found that software engineers have difficulty adopting new methods. They first learned to develop software during their formal educations and have since followed the same practices with a few adjustments and refinements. Since they are comfortable with these methods and have not seen compelling evidence that other methods work better, they are reluctant to try anything new. This problem is compounded by the fact that software engineers are rarely able to experiment. Everything they do is for delivery on a short and demanding schedule. An experiment would thus entail considerable risk. Not surprisingly, their reaction is to defer experimenting with new methods until they have some free time. Unfortunately, they never seem to have free time.

The current strategy is thus to introduce PSP methods in both industrial and academic environments with a formal course. In 15 weeks, the engineers develop ten small

programming exercises and write five reports. They analyze their exercise data and see where and how the PSP methods help them to improve. The course is demanding, however, and for industrial groups active management support is required along with job time to complete the exercises.

The SEI is offering teach-the-teachers training courses for industrial groups interested in introducing the PSP to their organizations. A textbook, *A Discipline for Software Engineering*, is also available from Addison Wesley.

For further information regarding S.P.I. Forum and for subscription information, contact Research Access <http>//www.rai.com/>.