

Applying the PSP in Industry

The Personal Software Process is designed for engineers working in isolated settings. The author describes a case study where he introduced the PSP in an industrial environment and the significant modifications needed to adapt it to a multiuser environment.

Maurizio Morisio, *Politecnico di Torino, Italy*

Despite the wide interest in the Personal Software Process,¹ the literature about its industrial applications is still limited. The existing PSP-related literature focuses on engineers working in isolated environments and their corresponding measures of improvement. It reports a correlation between the use of PSP by engineers in isolated settings (in a classroom or on single person projects) and improvements in defect density and estimation accuracy. Researchers haven't studied the effect

of other influencing variables. Although it would be good news to have similar advantages when using the PSP in real-life environments, with larger projects, larger teams, and normal schedule pressure, there is not enough evidence yet. (See the sidebar "Related Work" for more details.)

In the following case study, I introduced the PSP in a company and that effort raised the issues of training form and duration, resistance to change, independence and dependence between the single programmer and his or her team, overhead in data collection and tool support, and sustainability of measure. We addressed these problems by considerably modifying the PSP for that environment. One year after its introduction, only part of the PSP is still in use. The PSP proved to be more useful as a model to inspire a process improvement effort than as an off-the-shelf process to be reused without

modification. Although the modifications cannot be generalized, the experience gained can.

Experience Details

The PSP, which aims to improve the performance of individual developers, is based on three pillars—measurement, process, and techniques. The techniques for quality assurance (code and design reviews) and project management (size and effort estimation, planning, and scheduling) are the means to improve. Measurement demonstrates to the individual that his or her performance is improving. We need a process both to define when to use the techniques and to support measurement.

This case study took place at FIDIA, an ISO 9001 certified company in Italy that produces numerical controls and competes in the world NC market with large corporations. The company employs 250 people;

40 of them deal with software development or maintenance.

Although software accounts for a small part of an NC's cost and is even negligible compared to the cost of a machine tool, it is a strategic factor in NC production. The software can cause failures in the whole system, must be maintained for years, and can hinder the flexibility of the whole system in reacting to customers requests.

Software in an NC has three parts:

- a real-time embedded part that controls the axes of the machine in a path program's function;
- a monitoring part, with no real-time constraints and a sophisticated user interface to program, test, acquire, and analyze data from the first part; and
- the path program compiler. Another software program, starting from the high-level CAD description of the piece to be manufactured, produces the path program. This program is not real time or embedded, and it uses a complex user interface and sophisticated algorithms. The programming environment is C++ on Windows NT platforms. The program's size is approximately 140,000 lines of code—non-commented, source lines of code, excluding blank lines.

The PSP experimentation was on the third part, which a team of six software engineers develops and maintains. I worked as an external consultant, setting up the experimental framework and offering training and consulting about the PSP.

Why the PSP

The analysis of previous improvement efforts in FIDIA reveals that the pivotal point in the technology adoption was not the technology but people's acceptance of it. For instance, in experimenting code inspections, they observed that only the software reviewers who expressed interest in a method discovered errors, while the others, who appeared less interested or more skeptical, systematically obtained worse results.

Given this experience, we selected the PSP as the next improvement technology because we expected it

- to motivate software engineers to improve software quality;

Related Work

The corpus of literature about industrial applications of the PSP is still limited. Will Hayes and James Over analyze the measures coming from the Personal Software Process exercises performed by 298 engineers during their PSP training.¹ They do not provide information about engineers' experience levels or other demographics. The analysis of at least 170 data points shows that, on average, using the PSP, the engineers reduce the defect density in their work products by 30%, improve their effort estimation capability by 1.75 (median), and do not significantly change their productivity. In all cases, the data show a positive change in the engineers' performance through the PSP levels. However, these figures come from the isolated, ideal environment of a classroom, and we cannot extrapolate them directly into industrial environments.

Khaled El Emam and his colleagues remark that the high code reuse percentage (40% on average) between PSP exercises could be a factor influencing defect density and productivity.² They describe the introduction of PSP in a company where 28 engineers were trained using a modified PSP training—lectures on measurement and reviews and on-the-job practice. The engineers came from different teams in the company and chose on a voluntary basis to be trained. After seven months, 13 engineers were still using PSP concepts; 12 engineers could not apply the PSP for external causes, such as assignment to noncoding jobs and resigning. The main lessons learned are that PSP adaptation, tool support, and management backing are necessary to achieve success.

Pat Ferguson and her colleagues report about the use of the PSP in three companies.³ In company A, the estimation accuracy of engineers using the PSP on single-person small projects (up to 2,200 lines of code) improves considerably. In another set of three similarly sized projects (up to 30 requirements) developed by three engineers each—two using the PSP, one not—the PSP projects perform much better as far as defects found in requirements are considered. However, for defects found in usage during operational use by the customer, not by testers or developers, one PSP and one non-PSP project behave similarly well (zero or one defect found), and the other non-PSP project behaves much worse (14 defects). In the last set of two PSP projects (one engineer each, up to 2,200 LOCs), no defects are found in usage and acceptance test.

In company B, single engineers used the PSP on 18 projects, ranging from 1 to 4,500 LOCs. In all of them, except one, zero defects were found in usage. No figures are given for non-PSP projects, nor about the duration of usage.

In company C, the PSP was used in five maintenance and enhancement projects on the same product (one engineer, 200 to 6,000 LOCs). In the five cases, zero defects were found in usage. No figures are given for non-PSP projects.

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-97-TR-001, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, Dec. 1997.
2. K. El Emam, B. Shostak, and N.H. Madhavji, "Implementing Concepts from the Personal Software Process in an Industrial Setting," *Proc. Software Process 1996*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996.
3. P. Ferguson et al., "Results of Applying the Personal Software Process," *Computer*, Vol. 30, No. 5, May 1997, pp. 24–31.

We analyzed all potential failure factors and then modified the PSP to anticipate their occurrence.

- to provide them with means for continuous, individual assessments (instead of company-level exceptional assessments); and
- to give each software engineer a disciplined personal process, partially defined by himself and fitting his needs.

What Happened

Initially, the idea was to train the staff in the PSP, apply it as is, and modify it later, if needed. However, it soon became clear that this approach was not feasible. “Adapt the PSP to the company, not the company to the PSP” was one manager’s summary statement after the initial PSP presentation.

Training

The PSP packages not only a process to develop software but also a process to learn the PSP, which consists of lectures and seven to 10 exercises. The learning process has two goals: introducing the notions, techniques, and tools and inducing the student to get in the habit of following a disciplined process of logging and analyzing measures. As the measures demonstrate to the student her progress, she will be more willing to use the PSP and accept what, initially, seems to be an annoying overhead.

The SEI training oriented to industry users consists of two separate sessions of one week each. Besides the usual six to seven hours of training, each engineer is required to do two to three hours of additional homework per day between the sessions, in addition to another 15 to 20 hours of homework per week. Overall, effective training requires 150 to 200 hours,² or more than three weeks.

This is a true obstacle to PSP diffusion, not only because of the investment it requires, but also because it delays the normal workload of several weeks, especially in small companies and in small groups.

Others have attempted several workarounds:

- Self-training or training limited to lectures and exercises left as homework. This approach requires outstanding motivation and tenacity from the students.
- Reducing training and exercises, with the risk of students not reaching a PSP habit.

- Training a few people from the group each time. In this case, the training period can last months, at the group level, while the PSP habit decays.

The team manager and top management at FIDIA agreed that the team could not suspend normal work for three weeks, because the team supports the hot line (corrective maintenance and online help) for the path program generator. The team engineers, all with at least five years professional experience, were equally concerned about the loss of work time required to develop a number of student-level programs—what they considered to be boring and of little use.

They decided to train in three two-day sessions, with limited exercises. The sessions corresponded to three PSP levels (PSP0, PSP1, and PSP2) and were separated by experimentation periods of three to four months.

Introduction Process

The literature reports that most PSP technology transfers consist of training the engineers and letting them decide whether or not to use the PSP.³ Many of them stop using it after some months,⁴ and the FIDIA managers considered this unacceptable. Given the investment required, either the PSP was useful and used by the entire team, or they abandoned it. Instead, the FIDIA management tried to understand the causes of failure and to modify the PSP where needed. We analyzed all potential failure factors and then modified the PSP to anticipate their occurrence.

Independent Personal Processes

The PSP assumes a person working independently of his peers, capable of deciding and following his own lifecycle. However, this is not true in the FIDIA case. The high-level design of the product is well known to everybody in the group, and they discuss and solve design issues together. They also analyze and prioritize requirements for new features at the team level, based on informal input from customers and marketing people. In an ideal situation, a team would work full time adding new features to the product. In practice, customers point out failures the team must fix. Top priority failures always have precedence over anything else, because they can block manufacturing lines at the customer site.

When they add new features, two or more members together do requirements, design, integration, and testing; only coding, compiling, and testing are done in isolation. When defects are fixed, interactions with peers can be important too, and the lifecycle often reduces to a single phase.

As a consequence, we modified the PSP to take into account two different projects and lifecycles (new features and fixes), with activities shared by team members and with the possibility of working on several projects at a time. FIDIA redefined types of activities for effort logging, defect classes, coding standards, size measurement, and they are now the same for all team members.

Planning

A project is the unit of planning in the PSP—projects are done in sequence and are not related. In the FIDIA case, a team maintained and enhanced a product. A team member dedicated his daily time to enhancements and fixes, in varying proportions. Because a high-priority failure reported by a customer always interrupts any ongoing enhancement, they could not rigidly plan the proportion. Fixing a failure can require from a few minutes to days.

We adapted the PSP accordingly. New feature developments fit well in the concept of a project. However, a fix is too low-level to be modeled as a project; we modeled it as an activity with a duration for tracking purposes. We decided not to estimate effort on fixes, because the accuracy is too low and does not justify the time spent in doing it. However, we split fixes into two categories: quick fixes and those that require a large effort and reengineering. We treated the latter the same as new future development.

Quality Management

The PSP uses personal reviews on code and on design work-products. At FIDIA, design is mostly a team activity, so design reviews are at the team level, and they don't use personal design reviews. As suggested by the PSP, we introduced personal code reviews. We also kept the usage of peer code reviews in selected cases, especially when they developed complex algorithms.

The PSP considers a project until release, so measures end with the end of the project. At FIDIA, the team is in charge of a product

for its entire lifecycle. We revised the PSP quality measures accordingly to take into account all defects found in the product after release to the customer. They use these defects to characterize the quality of the product and process.

Team-Level Measures

FIDIA managers immediately realized that, if PSP measures were harmonized and aggregated at a team level, the team manager could have a powerful set of team measures. We introduced two levels of measures: the personal and the team level. Personal measures correspond, with some adaptations, to the PSP measures.⁵ We compute team-level measures by aggregating personal measures. They characterize both the team and the product they maintain. For instance, team-level measures available are the overall team effort, team effort by phase and by activity, and team effort per new features and fixes. Product-level measures available are size, defects found and fixed, and defect density.

In comparison with traditional measurement programs, at FIDIA, the data collection and analysis was not an add-on but was part of the everyday working style. This feature contributes to producing more reliable measures. However, tool support is essential.

Tool Support and Privacy of Data

The PSP introduces an important overhead for data collection and analysis and other activities. If engineers don't use tools, this means paperwork. Without tool support, the quality of data collected tends to be low.⁶ The engineers at FIDIA, all with an automation-oriented mindset, clearly stated that they could not accept such a level of paperwork.

Furthermore, some data requested by the PSP is already collected. The team collects data about effort spent on tasks and defects found and fixed. The FIDIA engineers immediately remarked the overlap between these activities and the same activities performed by the PSP, and they asked to avoid it.

As a result, we developed a tool. The tool supports the PSP but also substitutes any other tool or paper work previously used to collect effort, defect, and planning/tracking data, thus avoiding any redundant data collection.

If engineers don't use tools, this means paperwork. Without tool support, the quality of data collected tends to be low.

Introducing the PSP did not show a significant change in the trend of failures found by users.

It is well known that measurement programs can fail if measures are used to control people. The PSP collects data at a much higher level of detail, such as personal defect density and productivity. Therefore, the risk that a PSP initiative fails for this reason is much higher. In general, junior engineers are more concerned than senior engineers, whose performance is well known. Guaranteeing the anonymity of data is often not enough in a very small team, because we can easily trace some data (such as defects) to the author when the architecture of a product and the allocation of modules are well known.

The FIDIA developers realized immediately the threat to their privacy. An essential function of the tool is to support privacy of personal data, protected through passwords and cryptography, and to support the automatic computation of team-level measures.

Results

The FIDIA project started in April 1997, and the adaptation ended in April 1998. In September 1998, the project ended after several months of PSP usage. The results here consider measures collected through the PSP until September 1998, decisions by management, and opinions from PSP users. I collected the latter regularly through unstructured interviews during the project. In September 1999, I reassessed usage of the adapted PSP in FIDIA.

Estimation and Planning

Before the project, estimation at FIDIA was informal, based on analogy. The attempt to introduce the PSP estimation and planning techniques failed.

The main reason for this failure was the difficulty of making them work in a team environment. The team manager allocated the available team effort in priority to corrective maintenance and divided what remained between other activities (such as assistance to other teams and support to the commercial and marketing staff) and development of new product features. Even with a perfect effort estimation model for new features, a schedule is unstable, given the de facto priority given to other activities. Estimation and planning should start from corrective maintenance instead, but this is hardly predictable. Instead, the FIDIA team

argued that the best strategy was to focus on quality issues to reduce and ideally avoid corrective maintenance.

Another reason lies in the use of lines of code as a size measure. A couple of times during the project the team reengineered the product, resulting in important reduction of total LOCs. As a result, we could not establish meaningful relationship between LOCs and effort.

Effort Logging

At the beginning, engineers logged approximately 20 separate actions per day, with an average 25-minute duration and several interruptions. One year later, they logged half the number of actions per day, with an average 60-minute duration, and they no longer logged interruptions. When I asked about the change, they answered that recording interruptions was cumbersome, so they dropped it, and they partially reconstructed the way they spent time.

Eventually, they abandoned effort logging. This was a consequence of the burden of data logging, but also of abandoning PSP planning and estimation. Without them, the developers perceived no use for the data they were logging.

Quality Management

The PSP proposes code reviews and design reviews as the main tools to increase quality and several defect related measures to monitor it. FIDIA already used these tools at the team level, and they added personal reviews on code. Furthermore, the PSP helped improve tool support for defect logging and analysis of defect measures.

FIDIA deems failures found by users (customers or other teams in FIDIA) as the best indicator of the quality of the work performed by the team. They use failures still to be closed as an indicator of how well the team performs corrective maintenance. Because they consider all failures, a stable trend means that top priority failures are closed as needed. Defect density must be used with care because of the problems with size measurement.


Introducing the PSP did not show a significant change in the trend of failures found by users. This could be due to the fact that peer reviews on design and, partially, code reviews were already in place before the PSP.

The majority of the FIDIA developers were interested in the defect part of the PSP and especially in the possibility of monitoring the defect trend at the product level and trying to put it in relation with events of any kind—design choices, changes, release pressures. Therefore, the tool we developed to log and analyze defects and the habit of actually analyzing them are the main legacy of this PSP project.

From the point of view of the PSP, we could consider this project a failure. The developers did not acquire a PSP habit, did not change considerably their processes, and did not start monitoring themselves through the individual PSP measures. From the point of view of the team, the result is positive because there is now a better infrastructure for monitoring and analyzing defects and an improved measurement infrastructure in general.

If we consider investments and returns, the investment exceeds the returns. This is mainly because of the effort required to introduce and adapt PSP techniques that the team later abandoned. Considering only the parts that they still use, the return on investment is certainly positive.

The PSP appears as a packaged off-the-shelf process. The typical introduction model for the PSP consists of teaching it to engineers and letting them apply it as-is on their job. However, this model did not work in the case study I present here. We had to consider context variables, such as the interdependency among personal processes, techniques and procedures already used for quality and measurement, and human factors. As a result, we thoroughly modified the PSP.

Although it is risky to generalize from one case study, I believe the future of the PSP is not as an as-is process for industrial contexts. Researchers can use it as-is in academic teaching, where they can change context variables. Engineers can also use it as a process template to inspire team or group process improvement. Furthermore, new projects like IPSSI⁷ aim at developing evolutions of the PSP that are capable of building on its strengths while avoiding the weaknesses. 

Acknowledgments

The European Commission, DGIII/A, under contract Esprit/ESSI PSP-NC n. 24060, partially supported this work. Massimo Capra, Fabio Delton, Michela Oggioni, and all engineers at FIDIA offered invaluable patience and constructive ideas. David Escala participated in the PSP adaptation with essential concepts and dedicated work. Special thanks to Vic Basili for his help and encouragement and to Leonello Zaquini for his early support.

References

1. W.S. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, Reading, Mass., 1995.
2. W.S. Humphrey, "Using a Defined and Measured Personal Software Process," *IEEE Software*, Vol. 13, No. 3, May 1996, pp. 77–88.
3. P. Ferguson et al., "Results of Applying the Personal Software Process," *Computer*, Vol. 30, No. 5, May 1997, pp. 24–31.
4. K. El Emam, B. Shostak, and N.H. Madhavji, "Implementing Concepts from the Personal Software Process in an Industrial Setting," *Proc. Software Process 1996*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996.
5. D. Escala and M. Morisio, "A Metric Suite for a Team PSP," *Metrics 98*, IEEE Computer Society Press, Los Alamitos, Calif., 1998.
6. A.M. Disney and P.M. Johnson, "Investigating Data Quality Problems in the PSP," *Sixth Int'l Symp. Foundations of Software Engineering*, Orlando, FL, 1998.
7. G. Coleman et al., "Improving Individual Software Engineering Skills," *European Software Process Conf.*, (Euro SPI 200), Copenhagen, 2000.

About the Author



Maurizio Morisio is a research assistant at Politecnico di Torino, Turin, Italy. He recently spent two years working with the Experimental Software Engineering Group at the University of Maryland at College Park. His current research interests include experimental software engineering, software reuse metrics and models, the Personal Software Process, and commercial off-the-shelf processes and integration. He consults on improving software production through technology and processes. He received a PhD in software engineering and a MSc in electronic engineering from Politecnico di Torino. Contact him at the Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy; morisio@polito.it; <http://morserv.polito.it/morisio>.