PSP: Fair Warning

by Elizabeth Starrett
Air Force Mission Planning System

The Personal Software Process (PSP)SM Course is difficult and time consuming. I started working in a group that had already been trained on PSP and was using the Team Software Process (TSP)SM. I attended the PSP training with a second group from my section that had not had the training and was supposed to start implementing PSP and TSP on its upcoming project. This article discusses some of the difficulties the students experienced during and after taking the course. It also discusses the results experienced by my group already using TSP and the hope that life can get better.

So, you want to take a course on the Personal Software Process (PSP). Or, you do not want to take a course on PSP, but your management is forcing you. Either way, there are some things you should know before taking the course.

Definitions

PSP is a combination of personal software development processes as well as data collection and use suggestions intended to help the programmer develop software with better quality and predictability. TSP is a series of processes that incorporate the ideas of PSP into a team environment.

Prerequisites

First, make sure you are familiar with several basic functions that your development system provides. You will need to know how to code or call typical math functions such as square roots and natural logarithms. You will also need to be able to create, open, close, and access files; write to the computer monitor, and read from the keyboard. Make sure you know how to create and use linked lists. Using linked lists is a requirement dropped by some instructors who allow students to use arrays instead. Not all teachers are so lenient; mine was not. There is also a small group of people who prefer self-torture and will not use the equations provided by the teachers on blind faith. If you fall into this group and need to understand and believe the equations before using them, your preliminary knowledge will also require a basic understanding of statistics and its notation and calculus' concept of numerical approximation to integration.

Lots of Time

The next point—and probably the biggest—is the amount of time the course will take. There are different ways of formatting the course. The course is usually taught in two, one-week sessions. The first week of this format requires about 4.5 hours of lecture each day and the development of six programs, a coding standard, a Line of Code (LOC) counting standard, and a mid-term report. The second week requires about 4.5 hours of lecture each day as well as the development of four programs and a final report for the course.

If you are lucky enough to arrange a more lenient format, the homework will be the same, but the lecture time might be reduced to about three hours per day and there will be more days available to do the homework. My class was broken into four sessions. The first session was three days, the second and third sessions were four days, and the fourth session was three days (14 days vs. 10 days).

Maybe you are the brightest person and best programmer in your group. If you are, you can probably plan on attending the lectures and completing most of your programming homework in close to the same amount of time you would typically spend at work. You will probably need extra time for developing the coding standards, writing the sixth program, and writing the reports. One of my colleagues, whom I would put in this category, estimates he spent about eight total extra hours.

If you are one of the average programmers in your group, the amount of extra time you spend at work will depend on the amount of time your instructor spends on the lectures. Our lectures typically took about three hours each day. The time spent on just the 10 programs was an average of 62.6 hours per student. Add an average of 6.26 hours for each program to the lecture and you are putting in a lot of extra hours. Bear in mind that this documented time is only the task time and does not include times for interruptions such as taking phone calls, using the rest rooms, getting a drink, etc. You can typically plan on doubling the task time to get the overall time spent. Some programs are harder and some easier than others are, so the time you spend each day will vary. I was an average student and I spent about 30 extra hours working on the programs. However, I was not average on my time spent for the reports. My background is in the metrics arena, so I spent more than average time looking at all my data and trying to draw different conclusions for my reports. My mid-term report took about six total hours and the final report took about 20 total hours. (I also had 14 days to do this. If your course is taught in 10 days, the amount of extra time required will probably increase by about 32 hours.)

The programmer who has not been doing much programming lately or just is not that comfortable with it might want to avoid this class. Instructors have told me that they have had classes where at least one student spent months trying to develop the programs. The homework is not easy.

Implementation

After the course is over and the class has graduated, the process does not roll simply into the work environment. My class finished in November 1999, and four months later most of the students were still struggling to implement PSP and TSP into their newest project. My organization even has an internal tool that we developed to help automate the process, but the tool often creates as many headaches as the manual process. There have been numerous discussions on how work breakdown structures should be organized and how they should be used with the tool. Also, the tool merges with another SEI PSP tool and between the two of them, data is often lost and/or misrepresented.

Without TSP, PSP is not likely to be institutionalized. Six months after PSP was first taught in my organization, the instructor called all the students to see who was using it. Only one student was still using it. When the instructor called SEI asking for suggestions on how to institutionalize PSP, my current work group became a beta test site for TSP. The test results were impressive; both PSP and TSP are institutionalized in my group.

Results

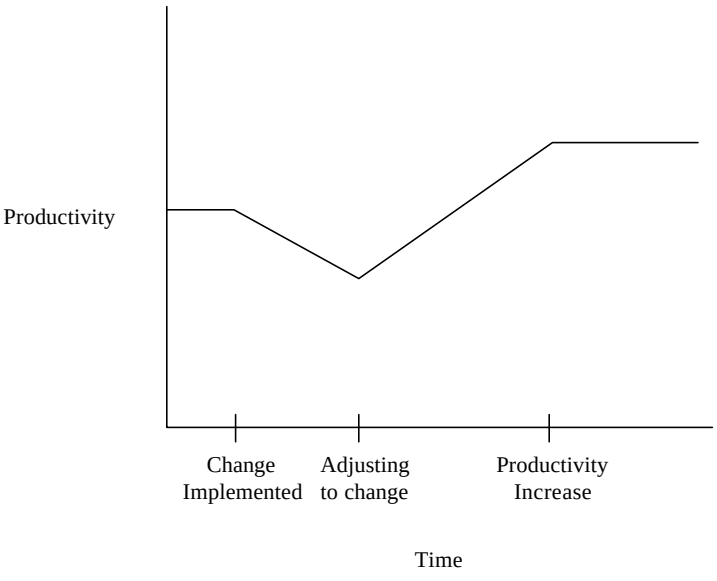
After all this struggle and all this work, do the PSP and TSP processes really help? Opinions vary. Several in the group that took the class with me agree the process portion of PSP is useful, but they do not agree all the data are useful and certainly not worth the effort to collect and try to use. This might be because they are not used to using the tools and will become accustomed to them with more time. Then again, they might not.

Figure 1 is a technology adoption curve that shows productivity decreases as any change is implemented. Learning a new way of doing things takes time. However, the theory is that a worthwhile change will cause productivity to increase to a point that overcomes the initial loss in productivity. Will the group now starting PSP make great and glorious gains? I do not know.

My work group is one of the first to implement the TSP and it received the division's quality award for the excellent results of the software released on budget, on schedule, and with minimal defects found in test. My group believes in using PSP, but opinions differ on the usefulness of some of the data. As time goes on, we may stop collecting some of the PSP measurements that we are not finding useful. My group agrees that the data collection process needs some sort of automation because a manual process is too cumbersome.

While the initial implementation of PSP is expensive, there is the hope that maintaining the PSP process will cost much less. Before coming to work in my group, I had already taken the PSP Executive Course (about eight hours) and understood the PSP concepts. I was able to walk into my current group and start using its process, with a few minor errors along the way. As I took the full PSP course, it became obvious that since I had already had the Executive Overview (so I understood why I did what I did) and I was already living the PSP process, I really did not need this course. I could have acquired the same knowledge with a brief course on the size- and time-estimating method and tool used by PSP (PROxy-Based Estimating [PROBE] method). After an organization's initial training and adoption into daily business, new people can be given the PSP executive overview, about two hours PROBE training, and a written process to follow to have what they need to use PSP with the rest of the group. This is assuming the rest of the group outnumbers the new people, so the experienced people can reasonably provide assistance. Maintenance has a much lower cost than the initial implementation.

Figure 1



Conclusion

I came to work in my current group after not developing software for 6.5 years. I came with an open mind and a desire to learn how to develop software using Level 5 processes. I was provided a written script of my group's software development process and support from my team. As a result, I have enjoyed developing code here much more than with my previous organizations and I have much more confidence in the code I am releasing to our customers. My opinion is that we are doing wonderful work, our customers are happy, and PSP is worthwhile; but I did not pay for it either.

Notes

1. The automated PSP tool developed and used by my organization is open source and freely available to anyone interested. The only payments requested are feedback from the users and the sharing of any improvements made to the software. The point of contact for further information on this tool is Ken Raisor (E-mail: ken.raisor@hill.af.mil).
2. There is a third group in my section that was already operating at a Level 5 before taking the PSP course. After taking the course, the group elected to continue using its old process instead of adopting PSP. However, the group's development processes are very similar to PSP.

About the Author

Elizabeth C. L. Starrett has been a member of the TaskView development team for the past year, supporting the Air Force's Mission Planning System. Prior to this, she was a software process improvement consultant for the Software Technology Support Center (STSC), where her duties included leading the STSC Measurement Team. Starrett has spoken at the Software Technology Conference, and at the Data Reduction and Computer Group Conference, and been previously published in CrossTalk. Prior to joining the STSC, she developed, documented, and tested data analysis and test support software for radar and the Peacekeeper missile, working for the Air Force and its supporting contractors. Starrett has a bachelor's degree in electrical engineering from Utah State University.

OO-ALC/TISHD
6137 Wardleigh Road
Hill AFB, Utah 84056
Voice: 801-775-2838 DSN 775-2838
Fax: 801-775-2541 DSN 775-2541
Internet: beth.starrett@hill.af.mil