TSP: Process Costs and Benefits

Jim McHale
Software Engineering Institute

The Team Software Process℠ (TSP℠), like other process improvement paradigms, is often challenged on the grounds that it adds overhead to already burdened developers. However, the TSP's overhead is readily quantified and justified by published results. One might also question the use of the term "overhead" when referring to necessary project tasks.

A question that often arises when attempting to convince a management team or an engineering staff to adopt the Team Software Process℠ (TSP℠), or any other process improvement program is: "How much overhead will it add?"

This is generally not an easy question to answer since it is unlikely that the questioner knows how much overhead, beyond perhaps a general accounting figure, is associated with an organization's current practices, particularly at the project-team level. Thus, the question of how much overhead will be added cannot be answered since it is not known how much overhead there was in the first place; a large part of that team-level overhead will likely be replaced by the TSP. Also, it seems somewhat ironic that this question arises from the same mindset that trims administrative and support staff, allowing managers and developers to answer their own phones and make their own copies, which seems like true overhead indeed.

For the sake of argument and part one of this article, let us suppose that the question is legitimate at face value. Part two of this article will address a few of the relevant benefits of using the TSP. Finally the question of process overhead itself will be examined.

Part One: The TSP Overhead

We need a counting standard to begin the analysis of the amount of the TSP overhead. We will assume 52 weeks per year, five working days in a week, and eight hours per day for a 40-hour week. That gives us 260 days or 2,080 hours per ideal developer-year.

The TSP calls for an all-hands team planning session (called a launch) that lasts for four days at the beginning of a project. Re-launches of up to three days each happen every three or four months. Let us assume that a full four-day launch happens annually, and a full three-day re-launch every quarter thereafter. This totals 13 days per developer, exactly 5 percent of the ideal developer-year.

At the end of every launch phase, the TSP calls for a post-mortem meeting to consolidate the data gathered and compare it against the launch estimates. The post-mortem also allows the team to figure out how the processes that they used helped or hindered in getting the job done, and to identify process adjustments to be implemented the next time. Post-mortems should last a day or less. Let us assume a full day for the entire team once a quarter, or four days per year. This is slightly more than 1.5 percent of a developer-year.

Weekly status meetings are also required. If the TSP team is reasonably efficient in running its meetings, most teams of, say, 10 to 12 people, can conduct them in an hour or less. Smaller teams can take less time, and larger teams can take a little longer, but if a meeting is running more than 90 minutes, either the team needs some training in meeting facilitation, or the team is just too big. Weekly meeting time equals one hour per week on average, or 2.5 percent of a developer-year.

> *"If you question the value of removing defects early via inspection, you should not even consider using the TSP or any other improvement paradigm based on CMM principles."*

The amount of time spent gathering data for the TSP is often questioned, so let us examine that. An entry in the time log, if done by hand on paper, takes about 15 seconds, counting start time, stop time, and interrupts as a single entry on one line. A person probably makes between 10 and 12 entries per day on average. It is a lot less tedious if you are using the SEI-supplied TSP tool or a freeware/shareware program, and there are some nice time-tracking programs available for your favorite hand-held device. Some organizations have developed their own tools for this purpose. [In case you were wondering, the Software Engineering Institute (SEI) does not care which tool you use, as long as you report specified summaries of the gathered data back to the SEI.]

If you are using anything besides a tool that allows direct consolidation with the rest of the team's data, it should take about five minutes to transfer your time log entries daily. (If you wait until the end of the week, it tends to be tedious and inaccurate. Do not do that!) Set transfer time at 10 minutes a day, 12 minutes to make the math easy. Five days times 12 minutes per day equals one hour per week. That is another 2.5 percent, but it is that high only if you do it by hand first and then transfer, less time otherwise.

Estimating the time spent logging defects can be tricky. Defects found in personal reviews or team inspections tend to take less time to log since, by definition, these are for things that you are looking for specifically. Set 30 seconds or less to log a defect found in reviews/inspections or by the compiler, which after all is telling you what the defect is. In integration and test phases, admittedly it can take a little longer to log a defect, but since you have relatively few of these (due to the great job you did in your reviews and inspections), the time spent here should not be too onerous. Even at two minutes per defect, that is plenty of time. On average, my informed guess is about one minute to log a defect. At 10,000 lines of code (LOC) per programmer-year (a fairly productive person) and an average of 100 defects per 1,000 LOC (KLOC), that is about 1,000 defects or about 1,000 minutes. To make the math easier, let us round up to 1,200 minutes or 20 hours per year, or slightly less than 1 percent overhead attributable to defect data gathering.

The time spent on the TSP role manager tasks is difficult to estimate, in part

because the actual duties of each role are very idiosyncratic to a particular team on a particular project in a particular organization. SEI guidance is for one to two hours per week. Two hours a week seems fairly high as an average weekly value, but even at that we are talking about another 5 percent overhead.

This analysis does not accept the premise that personal reviews or team inspections should be treated as overhead. If you question the value of removing defects early via inspection, you should not even consider using the TSP or any other improvement paradigm based on Capability Maturity Model® (CMM®) principles. CMM was developed as an instantiation of total quality management methods applied specifically to software development [1]. The basic tenet is that it is generally faster and cheaper to find defects earlier in the process rather than later. Also, if reviews and inspections are done properly, most defects should be found at that time and logged when they are fixed, and we certainly should not count defect-logging time twice. Therefore, in the TSP implementation of CMM principles, reviews and inspections are an integral part of the process, and not overhead.

The overhead numbers are summed up in Table 1. Is 17.5 percent a lot? I can't say. I contend that the question is irrelevant unless you know what you get in return.

Part Two: The TSP Upside

The most recently published example that I can find on the benefits of using the TSP is from a Honeywell presentation at the 2002 Software Engineering Process Group conference [2]. Pavlik and Riall claim a better-than-70-percent software productivity increase from one release of an avionics control system to the next, with a total of 22 percent savings in total systems and software effort.

While their delivery was on time, even more significant is that the quality of their delivered product was 10 times better than the previous release, while the delivered functionality was three times what was originally planned. I would say that a 17.5 percent process overhead for the TSP was more than worth it to Honeywell.

At the same conference, a presentation by John Ciurczak of EBS Dealing Resources claimed a "37.5 percent reduction in execution stage cycle" [3]. The execution stage, in this instance, refers to the time from when the business case for the project was approved, until the time that the product was tested and ready to ship. That reduction was entirely attributable to fewer defects in the EBS certification test phase, which is required for the foreign currency exchange services that EBS provides. Ciurczak also showed that the development activities prior to certification testing took just about as long with TSP practices as without. EBS probably cannot decide if that 17.5 percent was overhead, or just a normal and acceptable cost of doing a project.

In 2000, Don McAndrews of SEI published a summary of early results of using the TSP and its companion technology, the Personal Software Process℠ [4]. Unfortunately, productivity was not one of the numbers available for comparison. However, the before-and-after comparisons for cost and schedule deviation, defect density in test, and system test time per KLOC are usually cited by me and my SEI colleagues to ask a different question: "Can you afford not to implement TSP, overhead and all?" Certainly, the other numbers cited above leads one to the same question.

"TSP overhead performs necessary project functions regardless of the size of the organization ..."

Part Three: The Upside of Overhead

Finally, let us look at this accounting fiction called *overhead*. The battered dictionary that lives on my desk defines it this way: "*overhead* n.: business expenses not chargeable to a particular part of the work" [5].

I like this definition for a couple of reasons. First, it justifies my earlier tirade on not counting reviews and inspections as overhead, since one clearly must be inspecting a "particular part of the work." Second, it arguably removes the defect logging time from the calculations since that too can be attributed to particular parts of the job. The same argument applies for most of the time logging, since time usually is logged only against tasks that are traceable back to a specific part of what the TSP team is building. But for the continued sake of the argument, let us not fiddle with the 17.5 percent number, even though it may be high by a few percentage points. Let us talk instead about a subtle connotation of the word overhead.

Launches and re-launches	5.0%
Post-mortems	1.5%
Weekly team meetings	2.5%
Time logging	2.5%
Defect logging	1.0%
Role manager tasks	5.0%
Total	**17.5%**

Table 1: *Total Overhead Amounts*

In current usage, overhead conveys the sense that work so charged is somehow not entirely necessary. Is planning unnecessary? What about evaluating the effectiveness of the resulting plan during the last three months and figuring out how to do better the next time? What about knowing on a weekly basis just how effectively that plan is guiding the work, and how much of the work you have actually accomplished? What about gathering the data necessary to make these judgments about the plan? What about having the data available to evaluate your own performance objectively, and to deploy the team's resources most efficiently? These are exactly the purposes of the launches, post-mortems, weekly meetings, and data-gathering activities of the TSP.

Is it necessary for a project team to keep track of changing requirements? What about having common standards for design representation and coding? What about maintaining a view towards testing the system throughout the life cycle? These are all activities for some of the TSP role managers.

I have seen organizations create entire staffs to plan or to evaluate effectiveness, or to track time and defects, or to perform many of the other functions questioned above. While having such *overhead* positions may indeed be necessary in a particular organization, I like the TSP overhead model, which features the people who are doing the work also doing that little bit extra that actually helps them do the work more effectively and efficiently. TSP overhead performs necessary project functions regardless of the size of the organization, and can also help the people responsible for planning, evaluating, and tracking do their own jobs more effectively by freeing them to deal with the cross-project and other organizational issues that they are intended to address.

On balance, I prefer the overhead that the TSP brings to the table. A project team managing and measuring its own work against its own commitments, and getting results like those previously cited, seems to have no need to apologize for its practices or to justify what percentage of the

time they take to execute.◆

Acknowledgements

My thanks to Jim Porter of Tyco Electronics whose e-mail initially launched part of the preceding rant. My teammate on the TSP Initiative team at the SEI, Marsha Pomeroy-Huff, also provided the odd prod or two and edited my composition perhaps a little too gleefully. Finally, Anita Carleton of the SEI provided just the right push at just the right time to get me started, which is always the hardest part.

References

1. Paulk, Mark, B. Curtis, M. Chrissis, and C. Weber. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.
2. Pavlik, Rich, and C. Riall. Integrating PSP℠, TSP℠ and Six Sigma at Honeywell. Software Engineering Process Group 2002 Conference Proceedings (CD-ROM). Carnegie Mellon University, 2002.
3. Ciurczak, John. The Quiet Quality Revolution at EBS Dealing Resources, Inc. Software Engineering Process Group 2002 Conference Proceedings (CD-ROM). Carnegie Mellon University, 2002.
4. McAndrews, Donald. The Team Software Process (TSP): An Overview and Preliminary Results of Using Disciplined Practices. CMU/SEI-2000-TR-15. Carnegie Mellon University, 2000.
5. The New Merriam-Webster Pocket Dictionary. Simon and Schuster, 1971.

About the Author

Jim McHale joined the Software Engineering Institute in 1999. He has more than 20 years of experience, mainly in real-time control and supervisory systems in the transportation, steel, plastics, machine tool, and power generation industries. He has acted as software engineer, systems engineer, hardware engineer, project leader, and product manager. Since 1996, McHale has been a Capability Maturity Model® (CMM®)-Based Appraisal for Internal Process Improvement assessment team member, Software Engineering Process Group member, Personal Software Process instructor, and Team Software Process℠ (TSP℠) launch coach. Currently he is focusing on using the TSP to accelerate CMM-based improvement, and on adapting the TSP to enhance the effectiveness of other SEI initiatives, including commercial off-the-shelf systems. McHale has a bachelor's degree in electrical engineering from the University of Pittsburgh.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Ave.
Pittsburgh, PA 15213-3890
Phone: (412) 269-3948
E-mail: jdm@sei.cmu.edu

LETTER TO THE EDITOR

Dear CROSSTALK Editor,

I just wanted to thank the CROSSTALK staff and the authors for the April 2002 edition "Risky Requirements." I read every article with diligence. This is a very important issue that focuses on the most important piece of the software puzzle, software requirement.

Yet, in my opinion, it is the most neglected. You may have the best of everything: management, technical staff, resources, budget, schedule, customers, and even CMM® Level 5 processes. But, if you do not have a good set of well-defined validated requirements that are understood and agreed to by all stakeholders, you have absolutely NOTHING.

Did I say nothing? Well, you do have something. You have a whole lot of something: re-work, missed schedules, low quality, failed projects, irate management, customer dissatisfaction, canceled projects and additions to failure statistics.

We all know that there are no silver bullets for software development. But, if you have a good set of well-defined, validated requirements that all stakeholders can drive a stake into in the early stages of development, then that is the closest you will come to that silver bullet. Everything else will fall easily into place for the remainder of the life cycle. I know. I have experienced both phenomenons.

Al Florence
MITRE Corp.