The Team Software Process[SM] (TSP[SM]) in Practice: A Summary of Recent Results

Noopur Davis
Julia Mullaney

September 2003

TECHNICAL REPORT
CMU/SEI-2003-TR-014
ESC-TR-2003-014

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

The Team Software Process[SM] (TSP[SM]) in Practice: A Summary of Recent Results

CMU/SEI-2003-TR-014
ESC-TR-2003-014

Noopur Davis
Julia Mullaney

September 2003

Software Engineering Process Management Program

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Christos Scondras
Chief of Programs, XPK

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Acknowledgements

One of the principal components of the Team Software Process is building and sustaining cohesive teams. Our team at the SEI uses the TSP, and we feel we work with a great team. First, we must acknowledge our team leader, Jim Over. Jim gently reminds us of the commitments we made during our launches and keeps us from endlessly expanding the scope of our efforts. Jim always provides the right technical advice at the exact moment we need it. We also thank Jim for allowing us to spend three days in Knoxville to achieve a hundred percent earned value on this report. We thank Jim McHale, who always has a piece of candy ready just when we need it most, and who never loses his sense of humor, no matter how stressful things get. We thank Marsha Pomeroy-Huff for her editorial expertise, and no, Marsha, English is not our second language. We thank Dan Burton, who provided the most comprehensive feedback on the draft version of this report and caused us endless hours of discussions to resolve his insightful comments. We thank Anita Carleton, our data goddess, for setting a standard we will never live up to. If there are any data not properly presented in this report, it is entirely our fault and no reflection on Anita's comprehensive review. We thank Alan Willett for being an all around great team member and motivator, even though he didn't provide any feedback on this report until the very last minute. We thank Kim Campbell, who is responsible for gathering much of the data that was the input to this report. She is a great interface between the TSP project and the PSP/TSP community. Jodie Spielvogle best exemplifies being a TSP team member. She's the first one to step up and take responsibility, and she provided invaluable support in creating this report. Last but not least, we thank Watts Humphrey. Watts is a team member and so much more. Technically, Watts keeps our team all focused in the same direction. He provides the constancy of purpose required to support a long-term technology transition effort like this. He's a role model. He sets the ideal that we all strive to meet. However, we do not recommend using a stopwatch to measure interrupts when talking to your spouse; it doesn't work. But what we most admire about Watts is his innate belief in the goodness of people; that given the opportunity, people will do the right thing. We know this is a long paragraph, but we don't get many opportunities to thank our team.

Very special thanks to Jeannine Siviy, who led the development and administration of the survey on which some portions of this report are based. Jeannine also met with our community for discussion, analyzed and organized the survey results, established the SEI TSP data repository, worked with transition partners to understand their TSP data, and spent countless hours on initial versions of this report. We would also like to acknowledge Ken Smith for all his work with Jeannine.

Jim Herbsleb, from the Carnegie Mellon University School of Computer Science, and Jeff Roberts and Wai Fong Boh, from the Carnegie Mellon University Graduate School of Industrial Administration, developed Appendix A of this report. We thank them for their work in replicating the study originally conducted by Will Hayes and Jim Over in 1997, and for allowing us to use the results of the replicated study in this report.

The TSP does not work without supportive management. In this, we are lucky to have Bill Peterson, who encourages us to be a self-directed team. We also thank Bill for his review comments of this report.

Thanks to Darryl Davis for his insightful comments on the report. It is great to have expert advice available 24/7.

Pamela Curtis did an excellent job of providing us with editorial support. Thanks, Pamela.

Julia would like to acknowledge her father, Richard Gale, because during the throes of the final rewrite, she forgot to call him on his birthday.

And finally, we would like to thank the TSP community of coaches, instructors, team leaders, team members and managers. Thank you for helping us improve the method, and please keep sending us more data and more anecdotes about your experiences with the TSP.

Executive Summary

Most software organizations critically need better cost and schedule management, quality management, and cycle-time reduction. This report demonstrates that teams using the Team Software Process (TSP) meet these critical business needs by delivering essentially defect-free software on schedule and with better productivity.

The report starts with an overview of the TSP to provide the context for the results reported. These results include the benefits realized by a first-time TSP team, a summary of data from 20 TSP projects in 13 organizations, and stories from people who have used the TSP.

These TSP teams delivered their products an average of 6% later than they had planned. The schedule error for these teams ranged from 20% earlier than planned to 27% later than planned. This compares favorably with industry data that show over half of all software projects were more than 100% late or were cancelled. These TSP teams also improved their productivity by an average of 78%.

The teams met their schedules while producing products that had 10 to 100 times fewer defects than typical software products. They delivered software products with average quality levels of 5.2 sigma, or 60 defects per million parts (lines of code). In several instances, the products delivered were defect free.

The TSP improves schedule and quality management by creating an environment where individuals and teams can routinely do excellent work. This report concludes with stories and anecdotes that illustrate the personal rewards of using the TSP.

Abstract

Most software organizations are facing critical business needs for better cost and schedule management, effective quality management, and cycle-time reduction. The Team Software Process addresses these critical business needs. This report provides results and implementation data from projects and individuals that have adopted the TSP. The results show that TSP teams are delivering essentially defect-free software on schedule, while improving productivity. These data can be used for benchmarking and planning, motivation, lessons learned, and other guidance to those currently using the TSP or considering its use. The report also illustrates adoption experiences of practitioners in the field, including TSP team members, their managers, and their coaches and instructors.

1 Introduction

The success of organizations that produce software-intensive systems depends on well-managed software development processes. Implementing disciplined software methods, however, is often challenging. Organizations seem to know *what* they want their teams to be doing, but they struggle with *how* to do it. The Team Software Process[SM] (TSP[SM]), coupled with the Personal Software Process[SM] (PSP[SM]), was designed to provide both a strategy and a set of operational procedures for using disciplined software process methods at the individual and team levels. Organizations that have implemented the TSP and PSP have experienced significant improvements in the quality of their software systems and reduced schedule deviation [Ferguson 99, McAndrews 00].

The purpose of this report is to provide updated results on the use of the PSP and the TSP. The report starts with an overview of the PSP and the TSP to provide a context for the results reported. This is followed by a detailed description of the experiences of and benefits realized by a first-time TSP team. Often when TSP teams report their results, those results are met by skepticism. We hope that by walking through this first-time team's journey of using the TSP, we illustrate how the TSP creates an environment where skilled engineers can apply disciplined methods working on a cohesive and dedicated team. Next, we summarize the performance of more than 20 projects from 13 organizations that have used the PSP and the TSP. Then anecdotes from those in the PSP and TSP communities are presented to show how individuals view the PSP and the TSP. The report concludes with an appendix presenting the results of a study that reexamines the major hypothesis regarding the impact of the PSP on individual engineers first presented in a 1997 technical report [Hayes 97].

Practitioners and launch coaches will find that the data in this report provide useful guidance in implementing the TSP. For instance, the results provided can be used for benchmarking purposes. Benchmarks can be used by teams to set team goals and to identify their strengths and weaknesses. This report can also be used by people who are interested in using the TSP in their organization to show their management and teams the benefits of the TSP.

[SM] Personal Software Process, PSP, Team Software Process, and TSP are service marks of Carnegie Mellon University.

2 TSP Overview

The objective of the TSP is to create a team environment that supports disciplined individual work and builds and maintains a self-directed team. The TSP guides self-directed teams in addressing critical business needs of better cost and schedule management, effective quality management, and cycle-time reduction. It defines a whole product framework of customizable processes and an introduction strategy that includes building management sponsorship, training for managers and engineers, coaching, mentoring, and automated tool support.

The TSP can be used for all aspects of software development: requirements elicitation and definition, design, implementation, test, and maintenance. The TSP can support multidisciplinary teams that range in size from two engineers to over a hundred engineers. It can be used to develop various kinds of products, ranging from real-time embedded control systems to commercial desktop client-server applications.

The TSP builds on and enables the PSP. The PSP shows engineers how to measure their work and use that data to improve their performance. The PSP guides individual work. The TSP guides teamwork and creates an environment in which individuals can use the PSP to excel. Data from early pilots show that the TSP has been successful in addressing critical business needs [Ferguson 99, McAndrews 00].

2.1 History

In the 1980s, Watts Humphrey guided the development of the Capability Maturity Model® for Software (SW-CMM®). An early misperception of SW-CMM by some people was that it did not apply to small organizations or projects. In order to illustrate its application to small organizations, Humphrey took on the challenge to apply the SW-CMM to the smallest organization possible: an organization of a single individual. From 1989 to 1993, Humphrey wrote more than 60 programs and more than 25,000 lines of code (LOC). In developing these 60 programs, Humphrey used all of the applicable SW-CMM practices up through Level 5. He concluded that the management principles embodied in the SW-CMM were just as applicable to individual software engineers. The resulting process was the PSP. He subsequently worked on corporate and academic methods to train others to use the PSP technology.

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

As engineers started applying their PSP skills on the job, it was soon discovered that they needed a supportive environment that recognized and rewarded sound engineering methods. In many organizations, the projects in crisis receive all the attention. Projects and individuals who meet commitments and do not have quality problems often go unnoticed. Humphrey found that if managers do not provide a supportive environment and do not ask for and constructively use PSP data, engineers soon stop using the PSP. Humphrey then developed the Team Software Process to build and sustain effective teams.

2.2 What Makes PSP and TSP Work

Typical software projects are often late, over budget, of poor quality, and difficult to track. Engineers often have unrealistic schedules dictated to them and are kept in the dark as to the business objectives and customer needs. They are required to use imposed processes, tools, and standards, and often take shortcuts to meet schedule pressures. Very few teams can consistently be successful in this environment. As software systems get larger and more complex, these problems only get worse.

The best projects are an artful balance of conflicting forces. They must consider business needs, technical capability, and customer desires. Slighting any facet can jeopardize the success of the project. To balance these conflicting forces, teams must understand the complete context for their projects. This requires self-directed teams that

- understand business and product goals
- produce their own plans to address those goals
- make their own commitments
- direct their own projects
- consistently use the methods and processes that they select
- manage quality

Figure 1 illustrates how the PSP and TSP build and maintain self-directed teams. Successful self-directed teams require skilled and capable individual team members. Capable team members are critical because each instruction of a software module is handcrafted by an individual software engineer. The engineer's skills, discipline, and commitment govern the quality of that module and the schedule on which that module is produced. In turn, the modules come together to compose software products. Therefore, a software product is a team effort. The product's modules are designed, built, integrated, tested, and maintained by a team of software engineers whose skills, discipline, and commitment govern the success of the project.

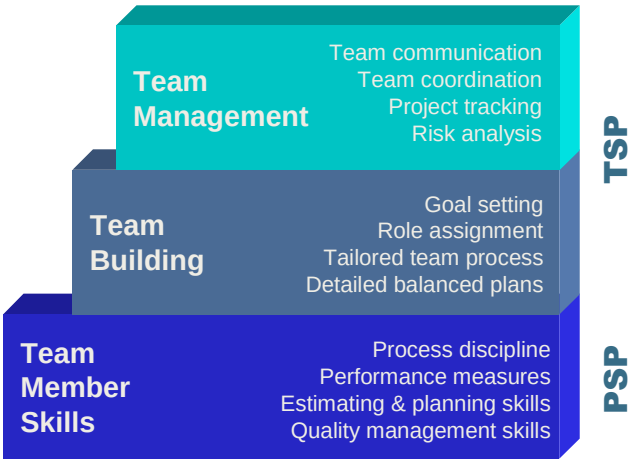


Figure 1: Elements of the PSP and the TSP

The objective of the PSP is to put software professionals in charge of their work and to make them feel personally responsible for the quality of the products they produce. The objectives of the TSP are to provide a team environment that supports PSP work and to build and maintain a self-directed team. PSP and TSP are powerful tools that provide the necessary skills, discipline, and commitment required for successful software projects.

2.3 The PSP

The PSP is based on the following planning and quality principles [Humphrey 00]:

- Every engineer is different; to be most effective, engineers must plan their work and they must base their plans on personal data.
- To consistently improve their performance, engineers must measure their work and use their results to improve.
- To produce quality products, engineers must feel personally responsible for the quality of their products. Superior products are not produced by accident; engineers must strive to do quality work.
- It costs less to find and fix defects earlier in a process than later.
- It is more efficient to prevent defects than to find and fix them.
- The right way is always the fastest and cheapest way to do a job.

Today, most software engineers do not plan and track their work, nor do they measure and manage product quality. This is not surprising, since engineers are neither trained in these disciplines nor required to use them. The dilemma is that until they try using disciplined methods, most software engineers do not believe that these methods will work for them. They won't try these methods without evidence, and they can't get the evidence without trying the methods. The PSP addresses this dilemma by putting an engineer in a course envi-

ronment to learn the methods. The engineers use the methods in the course and can see from their personal and class data that the methods can and do work for them.

The PSP course is composed of ten programming assignments and five reports. The PSP methods are introduced in six upwardly compatible steps, PSP0 through PSP 2.1 (see Figure 2). The engineers write one or two programs at each step and gather and analyze data on their work. Then they use their data and analyses to improve their work.

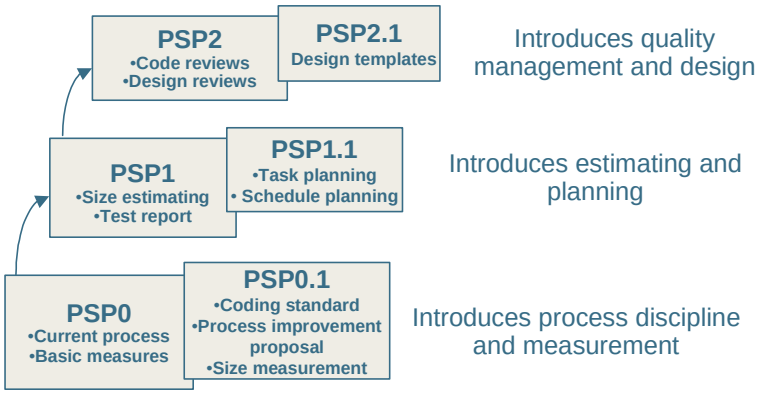


Figure 2: The PSP Course

PSP0 and PSP0.1. Engineers write three programming assignments using PSP0 and PSP0.1. The objective is for the engineer to learn how to follow a defined process and to gather basic size, time, and defect data.

PSP1 and PSP1.1. Once engineers have gathered some historical data, the focus moves to estimating and planning. Engineers write three programming assignments using PSP1 and PSP1.1. Engineers learn statistical methods for producing size and resource estimates, and use earned value for schedule planning and tracking.

PSP2 and PSP2.1. Once engineers have control of their plans and commitments, the focus of the course then changes to quality management. Engineers write four programming assignments using PSP2 and PSP2.1. Engineers learn early defect detection and removal methods and improved design practices.

Mid-term and final reports. After the first six assignments have been completed, engineers write mid-term reports, and after all ten programming assignments have been completed, engineers write final reports. These reports document the engineers' analyses of their performance. Engineers are required to analyze their data to understand their current performance, to define challenging yet realistic goals, and to identify the specific changes that they will make to achieve those goals.

By the end of the course, engineers are able to plan and control their personal work, define processes that best suit them, and consistently produce quality products on time and for planned costs.

In 1997, a study was conducted to analyze the impact of PSP training on 298 software engineers [Hayes 97]. This study found that engineers were able to significantly improve their estimating skills and the quality of the software products they produced. Engineers were able to achieve these notable improvements without negatively affecting their productivity. In terms of product quality and schedule variance, individuals were able to perform at a level that one would expect from a SW-CMM Level 5 organization.

The 1997 study was recently repeated on a much larger data set of over a thousand software engineers. The larger data set represents a more diverse group of instructors, engineers, programming languages, development environments, etc. The purpose of the replication was to demonstrate the statistically significant improvements in estimating and quality practices, i.e., to answer the question, can engineers learn to use their data to significantly improve their performance? The results from this replication are essentially the same as in the original study, with some minor differences. The findings are presented in Appendix A of this report.

2.3.1 PSP Measurement Framework

Engineers collect three basic measures: size, time, and defects. For the purposes of the PSP course, size is measured in lines of code (LOC). In practice, engineers use a size measure appropriate to the programming language and environment they are using; for example, number of database objects, number of use cases, number of classes, etc. In order to ensure that size is measured consistently, counting and coding standards are defined and used by each engineer. Derived measures that involve size, such as productivity or defect density, use new and changed LOC (N LOC) only. "New and changed LOC" is defined as lines of code that are added or modified; existing LOC is not included in the measure. Time is measured as the direct hours spent on each task. It does not include interrupt time. A defect is anything that detracts from the program's ability to completely and effectively meet the users' needs. A defect is an objective measure that engineers can identify, describe, and count.

Engineers use many other measures that are derived from these three basic measures. Both planned and actual data for all measures are gathered and recorded. Actual data are used to track and predict schedule and quality status. All data are archived to provide a personal historical repository for improving estimation accuracy and product quality. Derived measures include:

- estimation accuracy (size/time)
- prediction intervals (size/time)
- time in phase distribution

- defect injection distribution
- defect removal distribution
- productivity
- reuse percentage
- cost performance index
- planned value
- earned value
- predicted earned value
- defect density
- defect density by phase
- defect removal rate by phase
- defect removal leverage
- review rates
- process yield
- phase yield
- failure cost of quality (COQ)
- appraisal COQ
- appraisal/failure COQ ratio

2.4 The TSP

The TSP is based on the following principles:

- The engineers know the most about the job and can make the best plans.
- When engineers plan their own work, they are committed to the plan.
- Precise project tracking requires detailed plans and accurate data.
- Only the people doing the work can collect precise and accurate data.
- To minimize cycle time, the engineers must balance their workload.
- To maximize productivity, focus first on quality.

The TSP has two primary components: a team-building component and a team-working or management component. The team-building component of the TSP is the TSP launch, which puts the team in the challenging situation of developing their plan.

"Successful team-building programs typically expose a group to a challenging situation that requires cooperative behavior of the entire group [Morgan 93]. As the group's members learn to surmount this challenge, they generally form a close-knit and cohesive group. The TSP

follows these principles to mold development groups into self-directed teams. However, instead of using an artificial situation like rock climbing or white water rafting, it uses the team launch. The challenge in this case is to produce a detailed plan for a complex development job and then to negotiate the required schedule and resources with management."[1]

2.4.1 The TSP Launch

The first step in developing a team is to plan the work, which is done during the TSP launch. The launch is led by a qualified team coach. In a TSP launch, the team reaches a common understanding of the work and the approach they will take, produces a detailed plan to guide the work, and obtains management support for the plan. A TSP launch is composed of nine meetings over a four-day period, as shown in Figure 3.

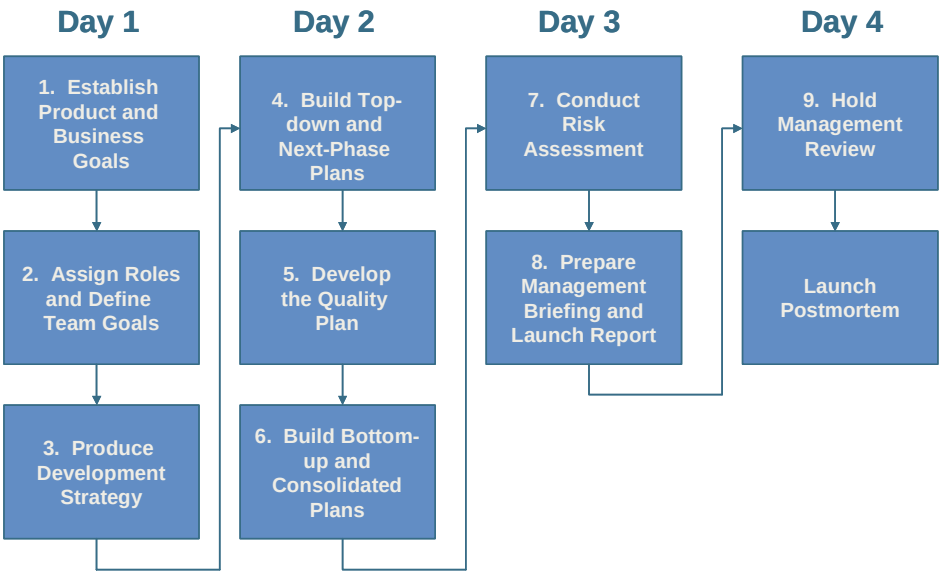


Figure 3: The TSP Launch

The first step in the launch is for the team to understand what they are being asked to do. This is accomplished in meeting 1 by having marketing (or an appropriate customer representative) and management meet with the team. Marketing describes the product needs. Management describes the business needs and any resources and constraints under which the team will have to work. This is also a chance for management to motivate the team. The team has the opportunity to ask any questions they might have about the product or business needs. In the next seven meetings, the team develops an engineering plan to meet the business needs.

In meeting 2, the team sets its goals and organizes itself. The team reviews the business and product goals presented in meeting 1, and derives a set of measurable team goals. Next, the

[1] Personal correspondence with Watts Humphrey.

team also decides which team members will take on which routine team management tasks. These tasks are designated by manager roles:

- customer interface manager
- design manager
- implementation manager
- test manager
- planning manager
- process manager
- support manager
- quality manager

Each team member selects at least one role. For teams with more than eight members, roles are shared. With smaller teams, team members may select multiple roles.

In launch meeting 3, the team determines its overall project strategy. The team members produce a conceptual design, devise the development strategy, define the detailed process they will use, and determine the support tools and facilities they will need. They list the products to be produced.

In meeting 4, the team develops the team plan. This is done by estimating the size of the products to be produced, identifying the general tasks needed to do the work and estimating their effort, defining the tasks for the next development cycle to a detailed work-step level, and drawing up a schedule of the team's availability week by week through the completion of the project.

In meeting 5, the team defines a plan to meet its quality goals. The team does this by estimating the number of defects injected and removed in each phase and then calculating the defect density of the final product. The team ensures that the tasks needed to achieve its quality goal are included in the team plan. The quality plan provides a measurable basis for tracking the quality of the work as it is done.

In meeting 6, tasks on the team plan for the next cycle of work are allocated to team members, and each team member creates an individual plan. In building their plans, the engineers refine the team estimates using their own historical data, break large tasks into smaller tasks to facilitate tracking, and refine their hours available per week to work on this project. The team meets again to review the individual task plans and to ensure that the work load is balanced. The individual plans are consolidated into a team plan. The team uses this plan to guide and track its work during the ensuing cycle.

The team conducts a risk assessment in meeting 7. Risks are identified and their likelihood and impact are assessed. The team defines mitigation and contingency plans for high-priority risks. Risks are documented in the team plan and assigned to team members for tracking.

Meeting 8 is used to develop a presentation of the team's plan to management. If the team's plan does not meet management goals, the team includes alternative plans that come closer to meeting management's goals. For instance, the team might be able to meet a schedule by adding resources to the team or by reducing the functionality delivered.

By the end of the launch, the team has formed a cohesive unit and created a plan that balances the needs of the business and customer with a feasible technical solution. The team has agreed on the technical solution that they propose to build and understands how that product will satisfy business and customer needs. The team agrees on the strategy and process for developing the product. The team has a detailed plan that it can use to guide and track the work. Team members all know who is responsible for which tasks and areas. Everyone on the team understands and agrees with the quality goal, and the team can monitor progress against that goal. Finally, the team has explored all of the things that might go wrong and has done its best to mitigate those risks. In short, the TSP launch provides a team with all of the conditions necessary to become a self-directed team.

In meeting 9, the team presents the plan to management for their approval to start the work. The team explains the plan, describes how it was produced (Figure 4), and demonstrates that all team members agree with and are committed to the plan. If the team has not met management's objectives, it presents one or more alternative plans. The principal reason for showing alternative plans is to provide management with options to consider in case the team's plan does not meet the organization's business needs.

Figure 4: The TSP Launch Products

At the end of the TSP launch, the team and management agree on how the team will proceed with the project. The team has a plan it believes in, is committed to, and can track against. The launch not only creates a winning plan, it builds a cohesive team.

The TSP includes guidance for ensuring that the energy and commitment from a TSP launch are sustained as the team does its work. A TSP coach works with the team and the team leader to help the team to collect and analyze data, follow the process defined by the team, track issues and risks, maintain the plan, track progress against goals (especially the team's quality goal), and report status to management.

2.4.2 TSP Measurement Framework

The TSP uses the same basic measures of the PSP—size, time, and defects—and adds task completion dates. For all measures, planned and actual data are collected at the individual level. The TSP measurement framework consolidates individual data into a team perspective. The data collected are analyzed weekly by the team to understand project status against schedule and quality goals. The TSP measurement framework also makes available other views of the data, such as by product or part, phase, task, week, day, etc. Personal and team data are archived to provide a repository of historical data for future use.

The team conducts weekly meetings to report progress against their plans and to discuss team issues. They also use their TSP data to make accurate status reports to management on a regular basis. Because management can rely on the data, their job changes from continuously checking project status to ensuring that there are no obstacles impeding the team's progress. This also allows management to make sound business decisions, since they are based on accurate engineering data. For example, when management is confident in the team's estimate, management can decide how to allocate resources to obtain a schedule that best meets the business needs. When a team commitment is in jeopardy, the team solves the problem or raises the issue with management as early as possible. In all cases and at all levels, decisions are made based on data.

2.4.3 The TSP Introduction Strategy

The SEI has been transitioning TSP into organizations since 1997 and has gained significant experience with issues surrounding the introduction of this technology. Based on these experiences, the SEI has defined an introduction strategy (Figure 5) and has developed supporting materials to facilitate the implementation of that strategy.

The introduction strategy starts with trial use. The TSP is first piloted on several small projects to evaluate both the transition approach and the impact of TSP on the organization. The pilots also build the understanding, sponsorship, and support needed for broad acceptance of the TSP in the organization.

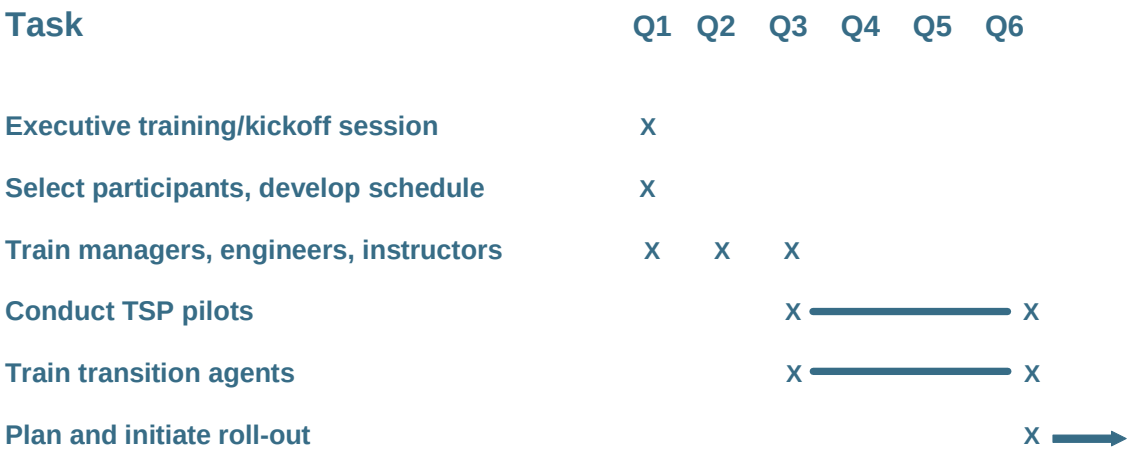


Figure 5: TSP Introduction Timeline

All team members and all of their management are trained prior to the start of the pilot effort. The senior management attends a one-and-a-half-day executive seminar and planning session; the middle and line management attend three days of training; the engineers complete the two-week PSP for Engineers course. The pilot teams are then started with a launch, and they begin to use the TSP process as they do project work. Pilot projects can rapidly demonstrate the benefits of using the TSP, and results from the pilot projects can be used to tailor and improve both the TSP and the introduction strategy.

3 TSP Results

The remainder of this technical report describes the usage and experiences that illustrate the benefits of TSP from three perspectives. We begin with a first-time project perspective. The experiences of a first-time TSP team are described in detail, from PSP training through product delivery. Next, we present some summarized project data. A summary of TSP data from thirteen organizations and at least twenty projects is presented. Finally, we switch from data to personal experiences. We present anecdotes from the PSP and TSP communities that relate experiences which cannot be described by data alone.

For each of these three perspectives, we first describe the source of the data. We then present the data. And finally we describe conclusions drawn from that data.

4 A First-Time TSP Project

4.1 Data Source

The team described in this section was part of a multinational company that has been involved with CMM-based software process improvement for several years. The organization worked with the SEI to use the standard TSP introduction strategy to pilot test the TSP on two projects. The data included in this section represents one of those pilot projects. We chose this project for several reasons. We have complete data on the project, from the PSP for Engineers course data, TSP launch data, and through to project completion. Also, we feel that this project's experiences typify what many first-time TSP teams encounter. In fact, this project was the first TSP pilot project to finish in this organization. And finally, this project encountered problems most software development projects do, such as effort underestimation and changing requirements.

4.2 PSP For Engineers Training Results

The class data presented here represents the engineers who completed most of the PSP for Engineers training. The data represents 84% of the 22 engineers who completed at least 9 of the 10 programs, and 4 of the 5 reports. The class included members of the pilot project described in this report, as well as other engineers. Throughout the PSP training, the key measures defined in the PSP Measurement Frameworks section (page 7) of this report are tracked for each of the 10 programs the engineers write. For each assignment, the class average and the minimum and maximum values for that measure are plotted. Don't assume that the maximum or minimum data point represents the same engineer. This is almost never the case. Engineers are presented with their class data throughout the course to illustrate the benefits the class is realizing from using these methods.

4.2.1 Time and Size Range

The charts in Figure 6 show the time spent developing each program in the course, as well as the size of that program.

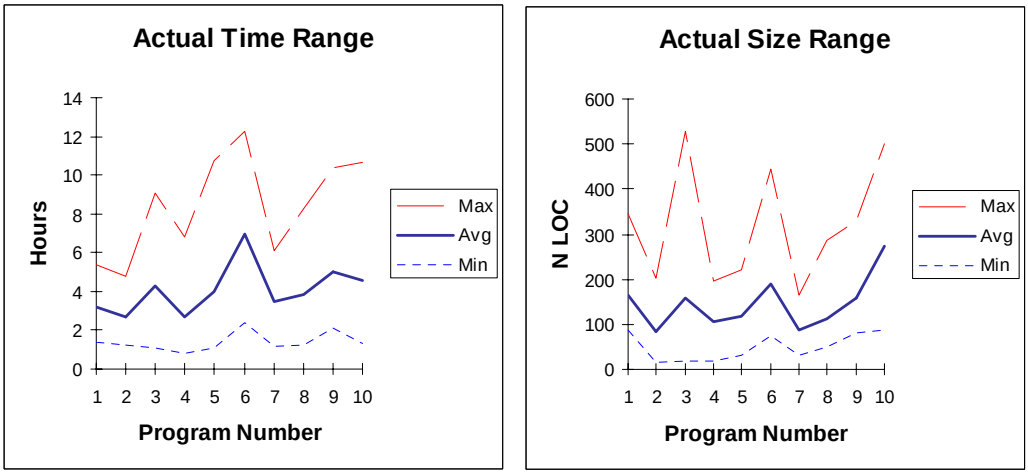


Figure 6: Time and Size Range

While the maximum and minimum time to develop each program varies widely, the class average is around four hours per assignment. Similarly the size range varies greatly between the maximum and minimum size. The size variation often represents different solutions for the same requirement. In this class, size variation was not due to differences in language or counting standards. All engineers used either C or C++ and the same counting standard.

4.2.2 Time and Size Estimating Accuracy

Figure 7 shows the time and size estimating error for each of the ten programs. The time and size estimation error is calculated as

%Estimate Error = 100 * (Actual – Estimate)/Estimate

Note that, unlike the rest of the charts in this section, the class line is a composite estimate for all engineers. For time, the composite estimate is the sum of the engineers' estimated hours. Similarly, the composite actual time is the sum of the engineers' actual hours. The class line is the error between the composite estimated hours and the composite actual hours. The class line for size estimating error is calculated similarly.

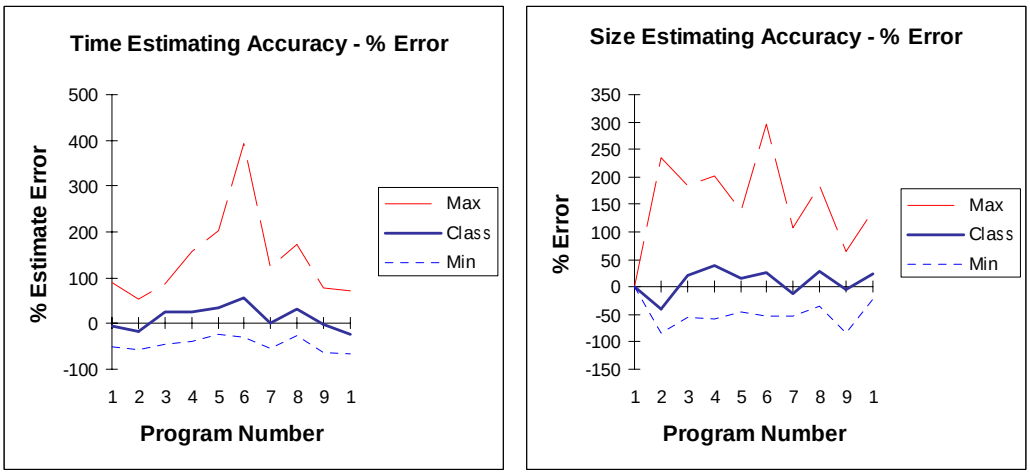


Figure 7: Time and Size Estimating Error Range

As can be seen, the composite estimation error appears to be stable, with an improving trend. This illustrates the benefits of independent, non-biased estimates. All engineers make non-biased estimates using statistical methods and historical data. Some engineers overestimate, while others underestimate, thus balancing the composite estimate. This helps engineers to understand the importance of breaking a problem down into parts and estimating each part independently using non-biased statistical methods.

Another point to note in the chart is the narrowing range of the estimation error. As engineers learn and use defined effort estimation procedures, the range between the maximum and minimum estimation error narrows.

As with the time estimate, the composite size estimate appears to be stable, with an improving trend, again illustrating the benefits of combining non-biased estimates. The range of the size estimation error narrows as the engineers learn to use a defined size estimation proce-

dure. (Note: Engineers did not estimate program size for program 1, hence the zero percent error for that data point.)

4.2.3 Compile

"Compile defect density" is the number of defects found by the compiler per thousand new and changed LOC. "Compile time" is the time it takes the engineer from when he or she starts compiling a program to when he or she gets a clean compile. The charts in Figure 8 show the defects found during compile and compile time as a percentage of total development time.

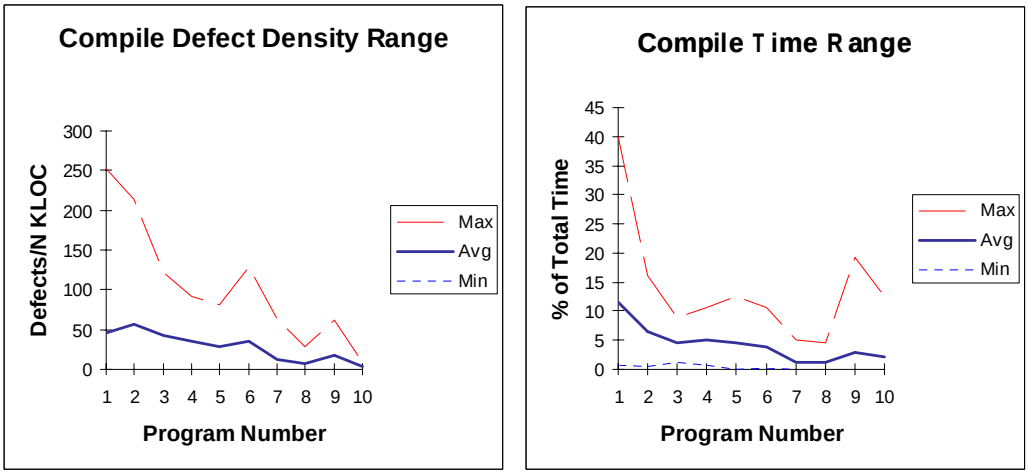


Figure 8: Compile Defect Density and Compile Time

In most development environments, compile errors are the only objective measure of the quality of the code. Reviews, inspections, and test defects can all be affected by the quality of the reviewers or the quality of the tests. Compile defect density is an early indicator of final product quality.

Engineers started the course spending about 9% of total development time removing about 51 defects per thousand lines of code (defects/KLOC) in compile.[2] After the introduction of personal code reviews and personal design reviews in program 7, engineers are able to remove most defects before compile. By the end of the course, engineers are spending less than 2% of total development time in compile, removing an average of 9 defects/KLOC. The quality of the code the engineers were compiling improved by a factor of five, thus time spent in the compile phase was minimal.

[2] See Section 4.2.7 for course summary data.

4.2.4 Design Time Range

In the PSP, design time includes the time spent reviewing the requirements and producing a design to meet them. The chart in Figure 9 shows the percentage of total development time that engineers spent in the design activities.

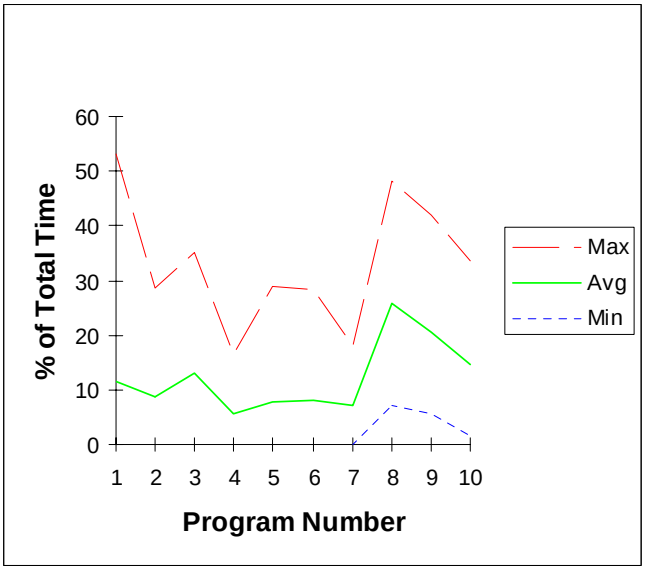


Figure 9: Design Time Range

Engineers use their existing methods through program 7. Starting with program 8, engineers are introduced to robust design specification methods. Engineers are surprised to see the small percentage of time spent in design activities prior to the introduction of these methods. For example, this class's data show that at the beginning of the course, engineers were spending as much time compiling their code as they were in designing their programs. Engineers claim to love design but seem most comfortable spending their time in compile and unit test (see Figure 8 and Figure 10).

4.2.5 Unit Test

Unit test defect density is the number of defects found during unit testing per thousand new and changed LOC. Unit test time is the time it takes the engineer from when he or she starts to test a program to when all tests run successfully. The charts in Figure 10 show the defects found during unit test and unit test time as a percentage of total development time.

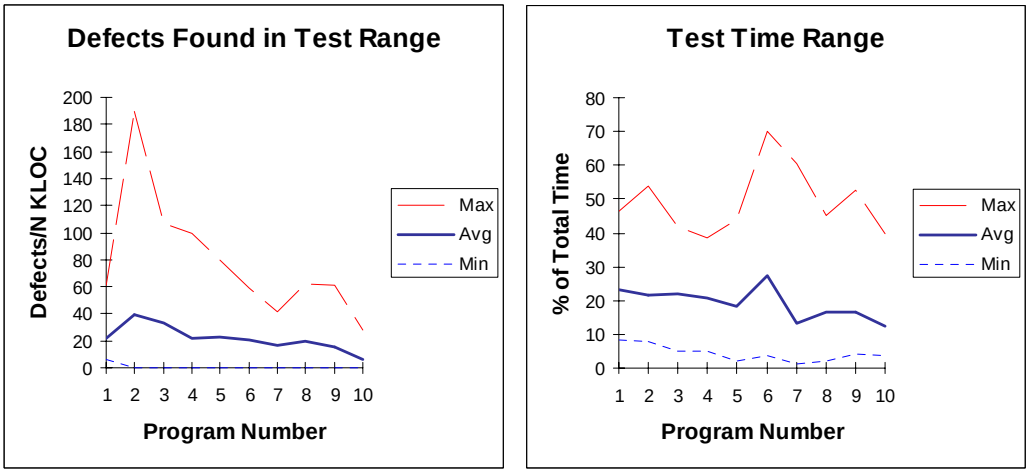


Figure 10: Unit Test Defect Density and Unit Test Time

Engineers started the course spending about 22% of total development time in unit test, removing about 31 defects/KLOC. With the introduction of quality methods in program 7 (personal design and code reviews and robust design specifications), these numbers are reduced. By the end of the course, engineers are spending less than 14% of total development time in unit test, removing an average of 8 defects/KLOC. Also, note the variance around the class average. The range between the maximum and minimum value narrows, but not because the minimum defect density worsens, but because the maximum defect density is reduced. This considerably improves predictability.

The quality of the code the engineers were testing improved by a factor of four (from 31 defects/KLOC to 8 defects/KLOC). Thus time spent in the unit test phase was minimized.

4.2.6 Productivity

Productivity is defined as development time per new and changed LOC. The chart in Figure 11 shows the change in productivity over the ten programming assignments.

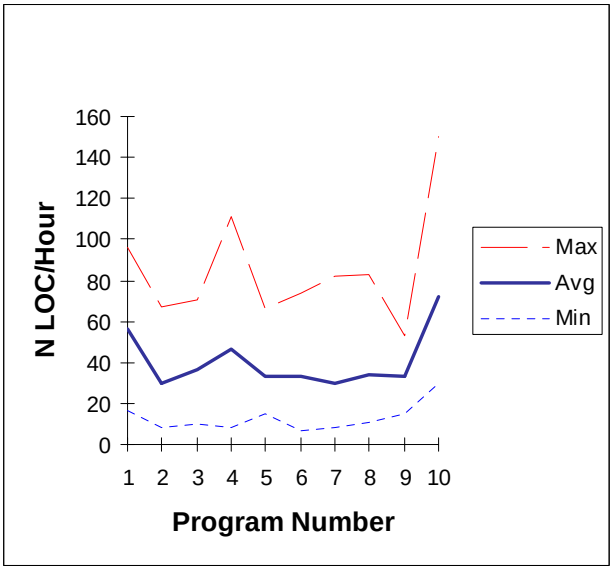


Figure 11: Productivity Range

An argument frequently made is that disciplined methods take too much time. These data clearly show that this is not the case for this team. There is no impact to productivity, yet the quality of the product entering unit test improved by a factor of four.

4.2.7 Class Summary

The results from PSP training were impressive, and consistent with results documented in Appendix A and in a previous SEI report [Hayes 97]. These results are summarized in Table 1. The first column describes the measure, the second column shows its value at the start of PSP training (class average for the first two programs), and the third column shows its value at the end of PSP training (class average for the last two programs).

Measure	At the start of training	At the end of training
Percent time spent in compile	9%	2%
Percent time spent in design	10%	17%
Percent time spent in unit test	23%	14%
Compile defect density (number of defects found during compile per thousand lines of code [KLOC])	51 defects/KLOC	9 defects/KLOC
Unit test defect density (number of defects found during unit test per KLOC)	31 defects/KLOC	8 defects/KLOC
Yield (percentage of defects found before first compile)	5%	55%
Productivity (detailed design through unit test)	43 LOC/hour	53 LOC/hour

Table 1: PSP Class Results

4.3 The Project Results

Less than a month after completing the PSP for Engineers training, the first TSP project was launched. The team consisted of five team members, including the team leader. The team leader was open-minded about the TSP and fully supported its use. Of the other four team members, one had fully embraced the PSP, another felt comfortable with using disciplined processes, and the other two were skeptical.

4.3.1 The Launch

Senior management was well prepared for the launch. Management discussed the importance of this product to the business, both from a functionality point of view and a sales potential point of view. Management told the team that their expectation was for the software to be delivered to the testing group in nine months.

A marketing representative followed the senior management presentation, and discussed the product needs with the team. The team members asked several questions, and launch meeting 1 ended on schedule.

The launch meetings proceeded in a fairly routine manner. The team members chose roles that best suited their abilities. Conceptual design took a while to complete. First, the team wanted to go into much more detail than was necessary at this point in the launch. After some attempts to get the team to step back and do the conceptual design at a very high level, the coach let the team get into more detail, knowing that the launch process was flexible enough to accommodate this. Next, the team questioned the need to define a development strategy, the products to be produced, and the development process for each product type. There was some impatience to get to what the team considered "real planning," or who does what task when. However, they agreed to follow the process and completed each step leading up to task identification, estimation, scheduling, and assignment. As the launch progressed, one team member told the launch coach that this was the first time he understood what commitments had been made on his behalf prior to the launch.

The plan the team came up with showed software delivery to test eleven weeks beyond management's schedule goal. Some team members were concerned about giving bad news to management. The launch coach wanted to make sure that the entire team believed in the plan, so the launch coach asked each team member if he or she thought the plan was too conservative, and why. Every team member agreed with the schedule the team had developed.

It was now about 7:00 p.m. on day three of the launch. The team had worked very hard for three days, and had not had dinner yet. They had completed the outline, as well as most of the content of the management presentation. The team leader and the launch coach volunteered to fill in the outline with the data captured on flip charts and artifacts during the earlier meetings of the launch, thus allowing the team members to go home. Not one single team member accepted this offer. This was their plan, and they were not going to let the team leader or the launch coach finish the management presentation. They stayed late and discussed each word and sentence until they were all satisfied the presentation represented their plan.

The next day, the team leader presented the plan to management. Management asked several questions and was especially concerned with the planned task hours per team member per week. In the end, management accepted the team plan and challenged the team to improve on the planned task hours per week.

During the launch postmortem, the team remarked on how surprised they were about the amount of work they had accomplished during the week. The team leader said the launch process was very good at guiding the team step by step through the planning process, thus allowing the team to concentrate on one thing at a time. She said that if she had known on day one everything the team would have to do during the launch, she would have been overwhelmed. The team members thought they should have been given more time before the launch to work on a conceptual design. They felt they were rushed during the size estimation

phase. Finally, the team members commented that having an experienced, neutral facilitator as a launch coach helped them to be successful.

4.3.2 Plan Summary

The plan the team developed during the launch is summarized in Table 2. The earned-value plan developed during the launch is shown in Figure 12.

Delivery to testing group	Week 47
Ready to release	Week 58
Effort estimate	2814 hours
New and changed LOC	14.5 KLOC
System test defect density	.36 defects/KLOC
Average task hours per team member per week	15 task hours/week

Table 2: Plan Summary

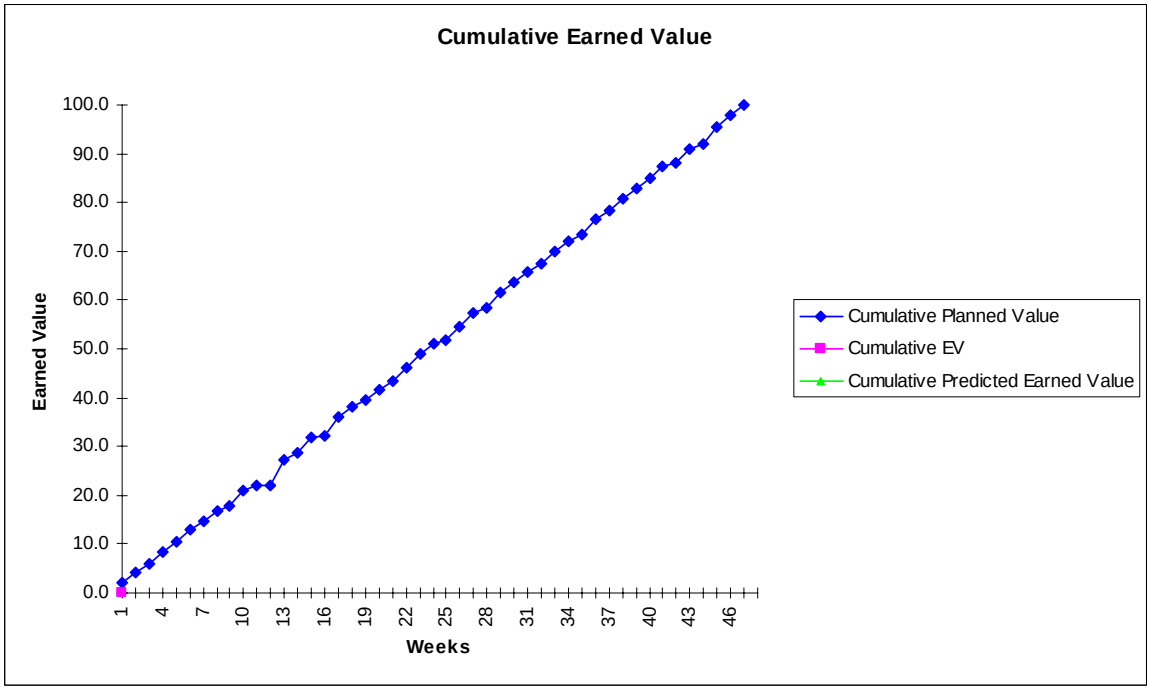


Figure 12: Earned-Value Plan

4.3.3 Executing The Plan

As with most software projects, this team encountered many obstacles as it started executing the plan it developed during the launch: some risks were realized, some tasks were underestimated, and requirements changed and grew. In order to simplify the story of this project,

snapshots of the project are presented at week 13, after week 46, after week 49, and after week 62. Week 13 was the week prior to the first relaunch, week 46 was the week before the original date the team committed to delivering the software to test, week 49 was when the team first delivered software to test, and week 62 was when the product was ready for release.

Week 13 – One Week Before Relaunch

A high-impact risk identified during the launch was that the engineers might be distracted from working on the new product because of the excessive support required for legacy products. This risk was realized. The team found itself providing more support than planned for an existing delayed release. By week 13, as shown in Table 3, the team was about 15% behind in task hours (plan of 650 project hours to date versus actual of 565.1), and had underestimated their work by 38% (plan of 306.6 for work completed versus an actual of 497.8). They had earned only half of the earned value that they had planned to earn. If the team continued to earn value at the current rate, the schedule exposure was at least six months.

Week 13 Data	Plan	Actual	Plan/ Actual
Project hours for this week	48.0	25.0	1.92
Project hours this cycle to date	650.0	565.1	1.15
Earned value for this week	1.8	3.3	0.54
Earned value this cycle to date	46.9	23.8	1.97
To-date hours for tasks completed	306.6	497.8	0.62

Table 3: Team Status at Week 13

In order to mitigate this schedule exposure, management decided to add three new team members to the project. During the launch, management had decided against additional resources, but when presented with data that showed the team was not going to make the schedule, management made a rational decision to add resources to the project. Therefore, in week 15, the team relaunched with the three new team members who had partially completed PSP training. During the relaunch, the team reestimated the remaining work. Based on lessons learned from the first 14 weeks and the additional overhead associated with the new team members, the team created a new plan. The new effort estimate was 3328 hours, an increase of 15% over the original plan. The release date was unchanged at week 58.

Week 46 – One Week Before Scheduled Delivery To Test

The team data at week 46 (Table 4) showed that the original team commitment for delivering the product to the testing group was 3% behind schedule (planned value of 96.5 versus an earned valued of 93.6). The earned value chart, shown in Figure 13, predicted completion by week 49, two weeks behind the date committed at the initial launch. The six-month delay that been predicted in week 13 had narrowed to only a two-week delay.

Week 46 Data	Plan	Actual	Plan/Actual
Project hours for this week	103.0	102.8	1.00
Project hours this cycle to date	3319.2	3829.8	0.87
Earned value for this week	1.7	1.0	1.75
Earned value this cycle to date	96.5	93.6	1.03
To-date hours for tasks completed	3051.7	3715.9	0.82

Table 4: Week 46 Status

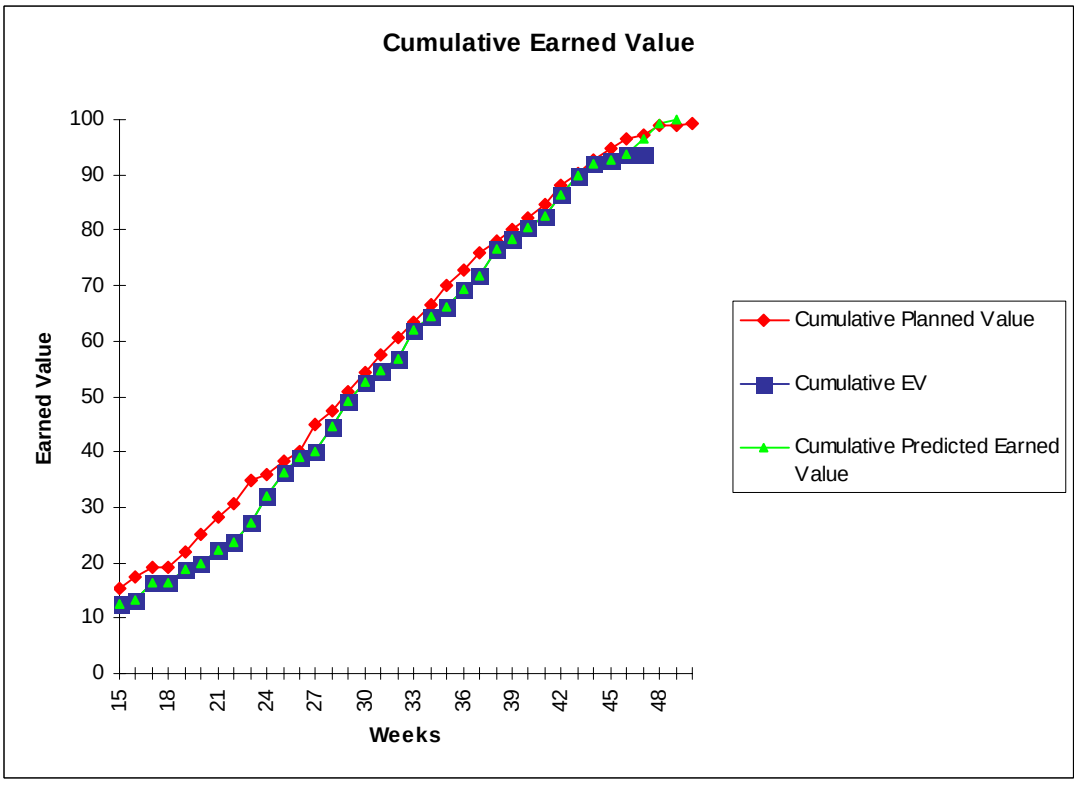


Figure 13: Earned Value Through Week 46

The team was able to close the schedule gap because they added three new team members, continually monitored their status and took corrective action as needed, replanned and rebalanced tasks as needed, and added a co-op student to help with non-critical tasks. The team

also improved average task hours per person, as shown in Figure 14 (week 18 was Christmas week, thus the zero task hours that week) and Table 5. The task hour improvement did not happen by accident: the team planned this during the relaunch. The task hours were not improved by working overtime; in fact, the team worked much less overtime on this project than on previous ones. The task hour improvement came by increasing uninterrupted time on task, by adopting quiet time, by streamlining necessary meetings and eliminating unnecessary ones, and by adopting flexible work hours. The interesting thing to notice is that just by task hour management, the team was able to increase its productivity by 28% (from 11.36 to 15.77 average task hours per team member per week).

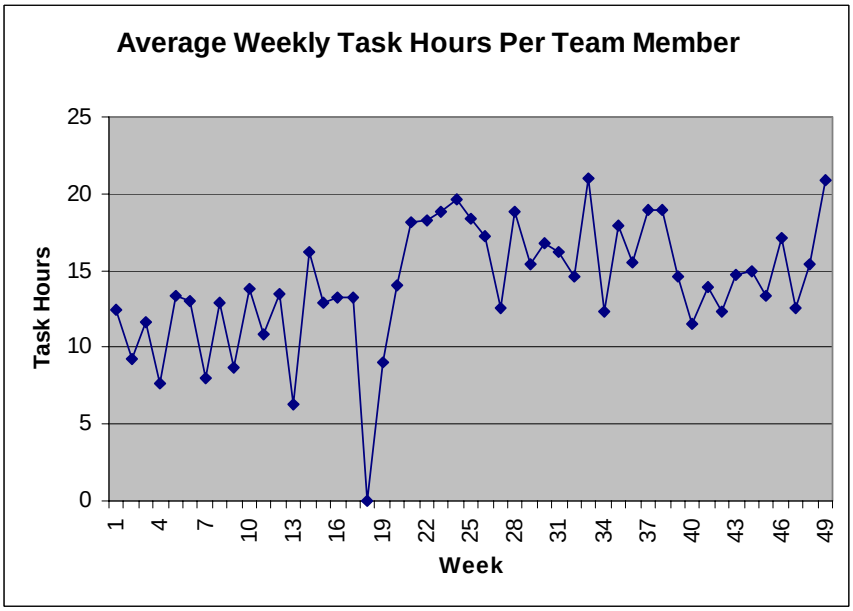


Figure 14: Average Weekly Task Hours

Average Task Hours Per Engineer Per Week	
Before relaunch (weeks 1-14)	11.36
After relaunch (weeks 15-49)	15.77

Table 5: Average Weekly Task Hour Improvement

Week 49 – Delivery To Test

The team was able to deliver software to the testing group in week 49, just two weeks behind schedule. The team was about a week away from delivering the software to the testing group when they were asked to implement a new requirement. The team estimated that it would take 252 task hours to develop the additional functionality. Management did not want the original release date of week 58 to slip due to the added functionality. The team was confident that the software they were delivering to system testing was of high quality. Therefore, they planned to spend very little time fixing system test defects. Using their historical aver-

age task hours, their plan showed that they could implement the additional functionality without delaying the final release date. They achieved this goal by planning to develop the additional functionality while the testing group was testing the initial functionality.

Week 62 – Project Completion

The team spent more than a third of their total development effort in quality activities such as design, reviews, and inspections (Figure 15). The software delivered to the testing group was thus of high quality. Before the software was delivered to system testing, 945 defects were removed, leaving less than 0.44 defects/KLOC to be found in system testing. Note that the team has an improvement opportunity for personal design and code reviews, but their inspections worked well (Figure 16).

As with typical software projects, testing proved to be an unpredictable activity. Even though very few defects were found during system testing, one defect was not discovered until a last-minute regression test. The product was released in week 62, four weeks beyond the date to which the team had committed over a year earlier, and with more functionality than they had originally planned. The final project status is shown in Table 6 and Table 7. Compared to a previous release of a similar product, these results represent

- 10x reduction in the number of problems logged by the testing group
- 8x reduction in system test duration

The team leader said the engineers were happy to have worked on a project that was delivered on time. She said it was a positive experience for everyone involved. The manager of the testing group said this was one of the most stable releases that his group had ever tested. The team members all said they enjoyed working on this project.

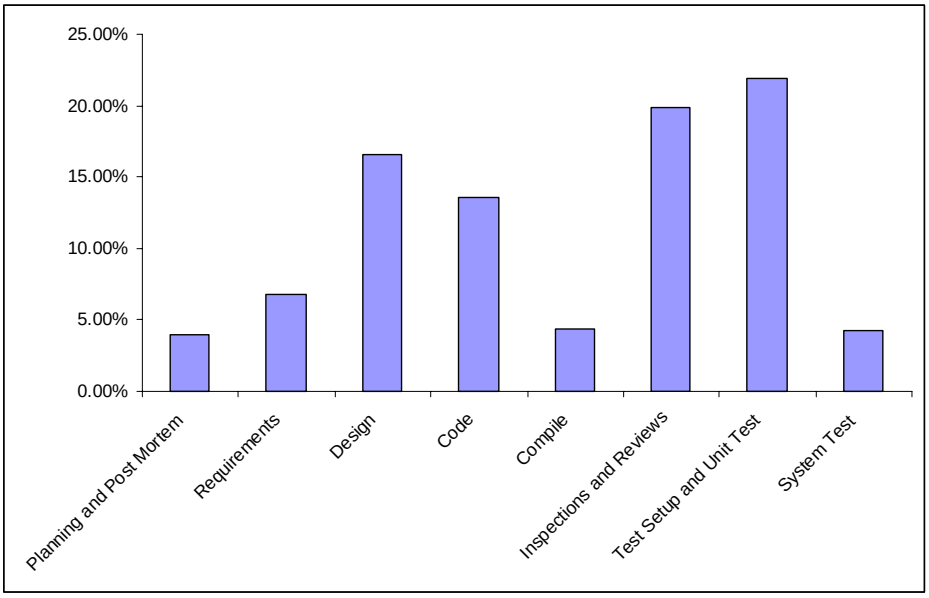


Figure 15: Effort Distribution

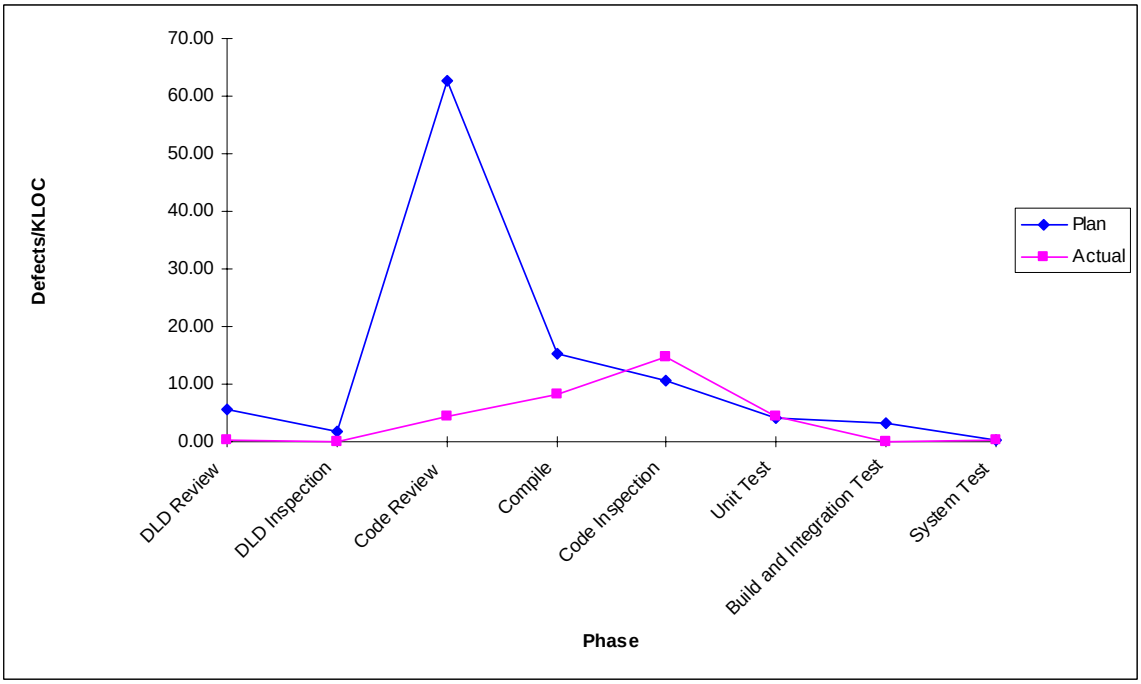


Figure 16: Defects Removed By Phase

Project length	62 weeks
Schedule variance	6.9%
Percent effort in system test	4.3%[3]
Defects removed before delivery to testing group	945

Table 6: Final Status

[3] Percent effort in system test is the effort the developers spent supporting the testing group (fixing defects found by the testing group).

	Plan	Actual
Delivery to testing group	Week 47	Week 49
Ready to release	Week 58	Week 62
Effort estimate[4]	2814 hours	3670 hours
New and changed LOC	14.5 KLOC	28.9 KLOC
System test defect density	.36 defects/KLOC	.44 defects/KLOC
Average task hours per team member per week	15	15.77

Table 7: Plan Vs. Actual

4.4 First-Time TSP Team – Conclusion

This project illustrated many of the problems that most software development projects face. For instance, the initial estimates were wrong, the requirements grew, and the team was constantly interrupted with work not directly related to the project. It's impossible to point to any one thing and say "this is why the team succeeded." First, the team has to know as early as possible when there is a problem. TSP teams collect data daily and review schedule and quality status weekly. This allowed this team to recognize in the early weeks of the project that they were falling behind schedule. Second, the team must understand what is causing the problem. In this case, the team had underestimated the work and overestimated available task hours. Then, the team must understand possible ways to address the problem. This team was able to take several steps to address the schedule slip. The team was able to provide management with the engineering data needed for management to make a business decision: should more resources be allocated to this project to bring in the schedule or should those resources be allocated elsewhere, letting the schedule slip? Management decided to allocate more resources to the project to maintain the original schedule. The team members analyzed their data and discovered that interruptions were preventing them from achieving their planned task hours. The team took active steps to correct that problem as well.

Another contributor to the team's ability to meet its schedule commitment was its focus on quality. The team made a quality plan and did not abandon that plan under schedule pressure. Because the team actively managed quality, they were able to produce a high-quality product and did not get caught up in endless test and fix cycles.

Many teams don't realize how much development time is "lost" while fixing defects that are found during system test and customer use. This time could instead be spent developing a new product or implementing new functionality. While typical software teams would be

[4] The plan and actual effort data does not reflect the additional functionality added in week 48 of the project.

spending an enormous amount of time fixing defects found in test, this team was implementing new functionality.

Finally, while the importance of using disciplined methods cannot be underestimated, it is the intangible, human elements of a TSP team that also determine whether a project will succeed or fail. This team was committed to their team goals, they depended on each other, they swapped tasks when needed, their team leader bought them doughnuts each time they submitted their weekly team data, they asked their team coach for help when their plans no longer worked for them, their management provided them help when they needed it, and they enjoyed working on a successful project. And so, while no single reason can be isolated for this project's success, it may be safe to conclude that the project may not have been successful without the combination of reasons described here.

5 Summarized Project Data

5.1 Data Source

The data summarized in this section come from all TSP presentations developed for the Software Engineering Process Group (SEPG) conferences (http://www.sei.cmu.edu/sepg) and the SEI Software Engineering Symposiums for the years 2001 through 2003 [Ciurczak 02, Davis 01, Janiszewski 01, Narayanan 02, Pracchia 03, Riall 02, Serrano 03, Schwalb 03, Sheshagiri 02, Webb 02].[5] We also examined the detailed data submitted to the SEI by the teams represented in those presentations (so launch coaches, please keep that data coming in!). The data presented here represent thirteen organizations and over twenty projects from these organizations. Some organizations presented summary data from more than one project without specifying the number of projects, so the exact number of projects could not be determined.

1. ABB, Inc.
2. Advanced Information Services
3. Bechtel
4. Cognizant Technology Solutions
5. Electronic Brokering Services (EBS) Dealing Resources, Inc.
6. Hill Air Force Base
7. Honeywell
8. Microsoft Corporation
9. Naval Air Warfare Center
10. Quarksoft, S.C.
11. SDRC
12. United Defense, LP
13. Xerox

[5] Also Ciurczak, John, "The Quiet Quality Revolution at EBS Dealing Resources, Inc.," Strickland, Keith, "The Road Less Traveled," and Webb, Dave, "Implementing the Team Software Process." Submitted for presentation at the Software Engineering Institute's Software Engineering Symposium, 2001.

5.2 Results

The data presented here are from a diverse group of organizations. Product size range is from 600 LOC to 110,000 LOC, team size range is from 4 team members to 47 team members, and project duration range is from a few months to a couple of years. Application types include real-time software, embedded software, IT software, client-server applications, and financial software, among others. Several programming languages and development environments were used (mostly third and fourth generation languages and development environments). We did not attempt to classify the data based on any of these differences. Instead, we gathered all the measures reported for each organization and calculated the range and average of the values reported. The ranges and averages do not include data from every project, as not all organizations reported the same measures.

We have also tried to compare the TSP projects presented here with typical projects in the software industry. This comparison is rather difficult to make, since there are not much data available on some of the measures tracked in the TSP. For schedule data, we used the Standish Group Chaos Report.[6] For time-in-phase data, we used several sources, including several estimation models, data from the NASA Software Engineering Laboratory [SEL 93], and pre-TSP data from some of the organizations we have worked with [Humphrey 02, Jones 95a, Jones 96, Jones 00]. For quality data, we mostly used Capers Jones as our source [Jones 95a, Jones 96, Jones 00], backed by pre-TSP data from some organizations we have worked with, as well as data from Watts Humphrey [Humphrey 02].

Jones uses function points as the size measure for normalizing defects (defects/function point). Since the TSP uses LOC as the default size measure, we had to convert function points to LOC. We used the "backfiring" method he described [Jones 95b] for this conversion. Jones suggests using a default of 80 LOC per function point for third-generation languages, and a default of 20 LOC per function point for fourth-generation languages. However, we chose to be conservative and used a default of 100 LOC per function point, as Jones does when discussing non-specific procedural languages.

5.2.1 Schedule Deviation

A premise of the TSP is to start with the best plan possible, using sound estimating and planning methods, and then update the plan as needed when you learn more about the work, or if the work itself changes. Because of the constant awareness of plan status, and because teams adjust their plans based on the plan status, TSP teams are able to reduce schedule error. The schedule data presented in Table 8 shows that TSP teams missed their schedule by an average of 6%.

[6] "CHAOS '94 – Charting the Seas of Information Technology." The Standish Group International, Inc., 1994.

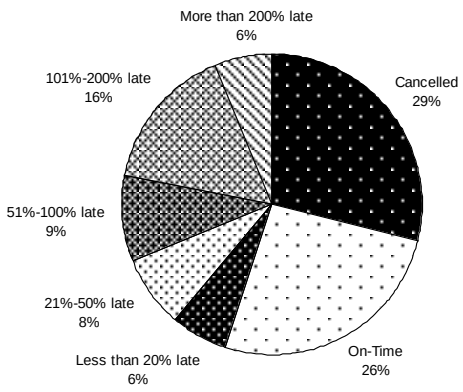
Measure	TSP Projects	Typical Projects (Standish Group Chaos Report)
Schedule error average	6%	
Schedule error range	-20% to 27%	

Table 8: Schedule Deviation

5.2.2 Quality

One reason TSP teams are able to meet their schedule commitment is that they plan for quality and deliver high-quality products to test. This shortens time spent in test, which is usually the most unpredictable activity in the entire development life cycle. The data in Table 9 show that TSP teams are delivering software that is more than two orders of magnitude better in quality than typical projects (0.06 defects/KLOC versus 7.5 defects/KLOC). Products being developed by TSP teams have an average of 0.4 defects/KLOC in system test, with several teams reporting no defects found in system test. TSP teams spent an average of 4% of their total effort in post-development test activities; the maximum effort that any team spent in test was 7%. Similarly, the average percentage of total schedule (project duration in calendar time) spent in post-development test activities was 18%. Typical non-TSP projects routinely spend 40% of development effort and schedule in post-development test activities. The 0.5 average days to test a thousand lines of code is a result of the higher quality of code entering system test. Some teams report that system test time was essentially equal to defect-free test time (time it takes to verify that the software works). Average failure COQ (percentage of total effort spent in failure activities) is much below the 50% typically found in the software industry.

Measure	TSP Projects *Average* *Range*	Typical Projects *Average*
System test defects (defects/KLOC)	0.4 0 to 0.9	15
Delivered defects (defects/KLOC)	0.06 0 to 0.2	7.5
System test effort (% of total effort)	4% 2% to 7%	40%
System test schedule (% of total duration)	18% 8% to 25%	40%
Duration of system test (days/KLOC)	0.5 0.2 to 0.8	NA[7]
Failure COQ	17% 4% to 38%	50%

Table 9: Quality

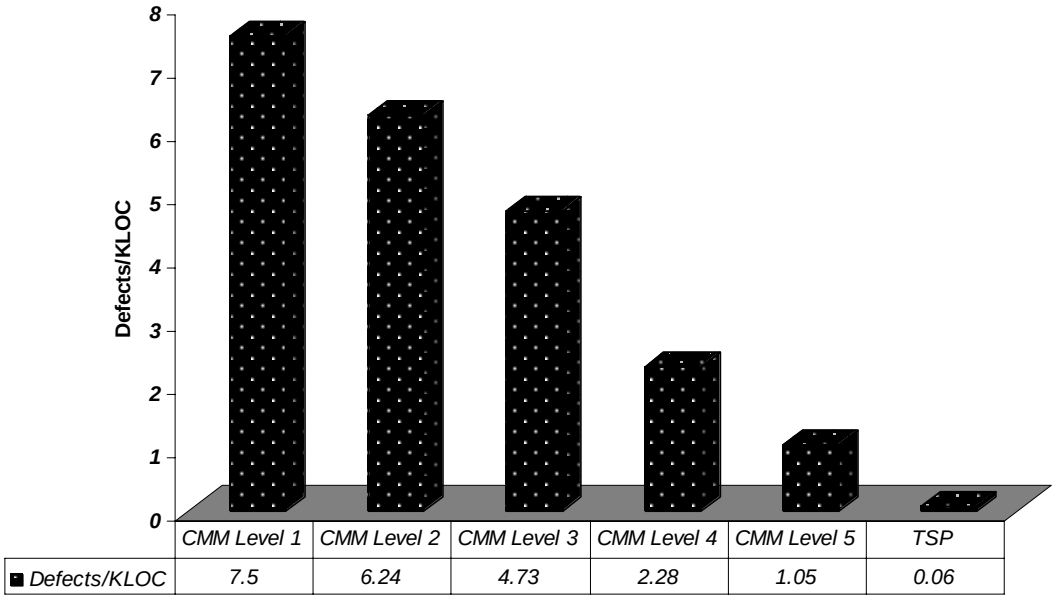


Figure 17: Average Defect Density of Delivered Software

Figure 17 shows the quality of delivered software classified by CMM Level [Jones 00], compared to the TSP teams presented in this report. These data show that TSP teams produced software an order of magnitude higher in quality than projects from organizations rated at CMM Level 5.

[7] This data was not available.

Some organizations reported the benefits of the TSP compared to previous projects (Table 10). They reported an average of 8 times reduction in system test defect density when using the TSP. System test duration was reduced an average of 4 times with the TSP: for example, a TSP project spending 0.5 days/KLOC in system test would have been spending 2.0 days/KLOC prior to using the TSP.

Measure	**TSP Projects** ***Average*** ***Range***
System test defect reduction	8 times 4 times to 10 times
System test duration reduction	4 times 2 times to 8 times

Table 10: Reductions In System Test Defects and System Test Duration

5.2.3 Quality is Free

A frequent concern expressed about disciplined methods is the perceived adverse impact on productivity. The data in Table 11 show that TSP projects improve their productivity and at the same time reduce their failure COQ (percentage of total effort spent in failure activities) and their total COQ (percentage of total effort spent in failure and appraisal activities). The main reason for this increase in productivity is the reduced time spent in test because of higher quality products being delivered into test, as shown in Table 9.

Measure	**Average**
Productivity improvement	78%
Failure COQ reduction	58%
Total COQ reduction	30%

Table 11: Improvements in Productivity and Cost Of Quality

5.2.4 Comparing Summary of Results

A previous technical report summarized TSP data from four organizations and fifteen projects [McAndrews 00]. Table 12 provides a comparison between the early results from TSP projects described in that report and the later results presented in this report. The later data represent a more diverse set of organizations than the earlier report (thirteen versus four organizations). We also have more complete data on the newer projects, and thus were able to calculate some measures that were not presented in the previous report. The data show that although the effort deviation range widened a little since the earlier TSP report, average schedule deviation remained basically unchanged. One conclusion that can be drawn from these data is that teams are able to manage effort deviation while they meet their schedule

commitments. The system test defect density, acceptance test defect density, and duration of system test show projects reporting even better quality results than those in the initial TSP report. The better quality may also account for projects meeting schedule commitments despite effort deviations.

Measure	**TSP Projects Results 2000**[8] ***Average*** ***Range***	**TSP Projects Results 2003**[9] ***Average*** ***Range***
Effort error	-4% -25% to +25%	26% 5% to 65%
Schedule error	5% -8% to +20%	6% -20% to 27%
System test defects (defects/KLOC)	NA 0 to 0.9	0.4 0 to 0.9
Acceptance test/released defects (defects/KLOC)	NA 0 to 0.35	0.06 0 to 0.2
Duration for system test (days/KLOC)	NA 0.1 to 1.1	0.5 0.2 to 0.8

Table 12: Results Comparison Between 2000 and 2003

5.3 Summarized Project Data – Conclusion

The results summarized in this section are remarkable when compared to typical software projects. The Standish Group reported in 1999 that 74% of all projects were not successful.[10] The Standish group also reported in 1996 that unsuccessful projects accounted for over half (53%) of total spending on software projects.[11] And in 1994, the same group reported that for the unsuccessful projects, the average cost overrun was 189% and the average time overrun was 222%. Typical projects spend 40% to 60% of total project time on test, and typical defect densities of delivered products range from 1 to 10 defects/KLOC [Humphrey 02].

As we reviewed data from a diverse group of organizations using the TSP, we were struck by the fact that to a large extent, these organizations were using a common operational definition for measures reported. For example, when projects report defect density, it is understood that they are talking about number of defects found per thousand lines *of new and changed* code only. Or when effort hours are reported, only on-task hours are measured. We also noticed a

8 Results from four organizations and fifteen projects reported in *The Team Software Process* [McAndrews 00].

9 Results from thirteen organizations and twenty projects presented in Section 5 of this report.

10 "CHAOS: A Recipe for Success. Project Resolution: The 5-Year View." The Standish Group International, Inc., 1999.

11 "CHAOS '97 – The Changing Tide." A Standish Group Research Note. The Standish Group International, Inc., 1997.

common language used for project management: terms such as yield, cost of quality, earned value, task hours, and defect density all have the same meaning across projects and across organizations. Common operational definitions of measures, as well as a common project management language, are both results of using the TSP. These results show that the team described in Section 4 is not unique—all of these projects were able to overcome similar obstacles and be successful.

6 Anecdotes

While quantitative data are important, numbers illustrate only part of the results. Equally important are the stories behind the results, both positive and negative. In this section, we provide some of the stories and comments behind the data. Positive stories illustrate the qualitative benefits TSP teams have been able to achieve, while negative stories provide valuable lessons learned.

6.1 Data Source

The stories and comments in this section of the report come from four primary sources. The first source is a survey that was sent to the SEI authorized PSP instructor and TSP launch coach communities asking about their experiences with PSP and TSP.[12] The second source is the evaluation forms and reports completed after each launch or relaunch and submitted to the SEI. The launch coach and all team members complete evaluation forms. The launch coach often writes a launch report to document significant events from the launch. The third source is the project post mortems conducted at project cycle completion or at project completion. This post-mortem data is also submitted to the SEI. The fourth source is presentations developed for the Software Engineering Process Group (SEPG) conferences and the SEI Software Engineering Symposiums.

6.2 Anecdotal Results

Most anecdotes presented in this section are verbatim from the individual; however, we have changed proper names and gender, and corrected grammatical and spelling errors. Some anecdotes are synopses of stories related to our team either verbally or through launch reports.

[12] A summary of the survey questions is included in Appendix B. Appendix B also contains detailed information about the people who replied to the survey; e.g., whether they are still using PSP/TSP, what kind of software their group develops.

6.2.1 Introduction Strategy

The introduction strategy developed by the SEI incorporated lessons learned from many projects we have worked with. This section shows similar lessons learned by other organizations about the introduction strategy as described in Section 2.4.3, page 12.

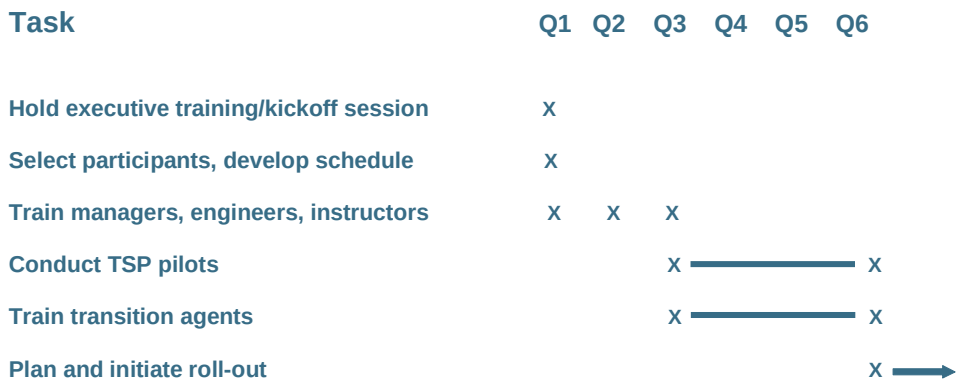


Figure 18: Introduction Strategy and Timeline

"Don't do TSP without complete PSP training."

"It's better to train the engineers who are going to be involved in the TSP pilot projects."

"The majority of people do not use PSP unless they are part of a TSP team—even though they loved the course."

"My personal experience is implementing PSP alone will not yield benefits. You should implement PSP with TSP. Then only do you get excellent benefits."

"After training the engineers with PSP, have TSP launches as soon as possible. When there is a gap, people tend to go back to the original process if left unattended."

"Don't try to do all software teams at once. Get a small pilot started and into system test. Then use their success to sell this, sell this, sell this."

"Most pilots that fail never make it to a launch. They fail because of shifting management priorities or lack of sponsorship in training. The other road to failure is making special exceptions for a cowboy. Anyone that is not prepared to do the process and seriously try to make it work should be removed from a pilot project ASAP. Anyone that proactively undermines an organization's pilot project should be removed."

6.2.2 Conviction to Change

PSP for Engineers training is much more than skill training: the purpose of the PSP for Engineers course is to provide people with the conviction and motivation to change the way they work. The course is very personal. It is about collecting *your* data, to understand *your* performance, so you can improve what *you* are doing, and to strive to reach *your* personal best. The anecdotes in this section are typical of what we hear in PSP for Engineers classes. TSP teams foster an environment that nurtures and supports personal improvement.

"The course is not about the lectures. The course is not about the programming the students do—how fast/well they do it. The course is about the students seeing for themselves how they really work, and realizing that they can make improvements. If students see it as something for them, then they are much more likely to carry it on afterwards."

"The best part about PSP/TSP is that collecting the metrics is for my benefit, not for someone else. I found that collecting the data proved to me that using a better process really does help my quality and productivity."

"In the end, what this is really about is people. No matter what you are investing in, what training, process improvement effort…what you are really investing in is people. And the important thing is that we improve what we are doing."

"A student related this story to me about his own 'conversion.' He had been doing well in class. We were already late in the course, and had our code review and design review checklists and were using them. This student was doing the course in Object Pascal. He got called in to do some work with a customer on a program he had helped with several years ago. He was supposed to add some functionality. He and two other programmers were working on it. He said, 'I considered taking my review checklists with me but thought, nah, my review checklists are for Pascal, not for ADA.' So he and the others worked on the project. They got to the end of coding and tested. There was something wrong in test—it didn't work. It took them 6 hours to find the defect (18 man hours). When they finally found it, it was a missing ';'. This was one of the things my student had on his checklist. He was a very strong proponent of PSP after that."

"In week 1, engineers complain endlessly about why they have to collect compile defect data. In week 2, engineers complain about 'that one compile defect I should have found in my review!' Engineers have the conviction and motivation to produce high-quality software. Their attitude fundamentally changes."

"Although I have had some negative experiences, on the whole, being involved in the PSP/TSP has been very positive. I have seen whole teams turn from bad to great in terms of quality, productivity, and morale. THIS STUFF CHANGES PEOPLE'S LIVES. I know that's a little heavy-handed, but I think if it is properly implemented, TSP can become a way of life for software engineers; of the six members of our original TSP team, three of us are now TSP launch coaches. And all of us find ourselves using it even in our private lives. Pretty impressive, huh?"

"It is so revolutionary that I remember the exact date I was introduced to it."

"A more disciplined process allowed me to do a better job, and allowed me to balance my job with other aspects of my life."

"Gives you incredible insight into personal performance."

"I thought I was a great programmer, and had been told so throughout my career. When I read the Discipline for Software Engineering[13] *book, I thought the data presented in it was typical of average developers and mine would be much better. The course was a humbling experience. I learned that I was as average as anyone else."*

"At the end of the class, we were talking about what we had learned. One of the engineers stood up and said "I don't have twenty years of programming experience, I have one year of experience twenty times. Until this class I did not even know what my performance was."

[13] [Humphrey 95]

6.2.3 Realistic Plans/Schedule Performance

As we saw in the team described in Section 4.3, the team met their schedule commitment. The team had the skills to put together a very detailed plan. And they had the skills to change and adjust that plan when the plan was not working. They did not abandon the plan; they changed it and constantly used it to guide their work. Therefore, they always knew where they stood against their schedule goal.

This section focuses on how engineers are able to apply the planning and estimating skills they learn in the PSP for Engineers course to meet schedule goals on TSP teams (see Figure 19).[14]

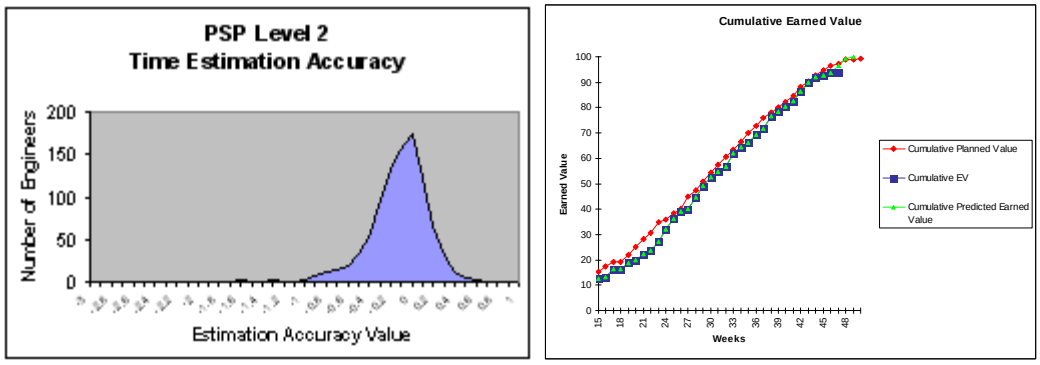


Figure 19: Realistic PSP and TSP Plans

"Probably the most impressive thing to me about TSP was the time that another team requested the services of one of my engineers. My first impulse was to say, 'No way, I won't get my earned value!' However, I decided to listen to the data and see what we could do. Because of the detailed plans made by TSP teams, I was able to sit down with my engineer, zero out his time for the six weeks the other project wanted him, look at what happened to his earned value and how that affected the team earned value. I was then able to look at the completion rate of the other engineers and find an engineer who was ahead of schedule. Working with those two, we were able to determine what tasks could be swapped from one person to another and recalculate earned value based on the changes. With the tasks moved and EV recomputed, my project was still projected to meet schedule, so we agreed to the engineer working for the other team for six weeks. This had three major effects: (1) it taught us that data could be used for a very practical application, (2) it reinforced the team's faith in me as a team leader, since I did not make a gut decision but used data, (3) it improved team morale, since I was flexible enough to allow the team members personal freedom as long as their data supported it. By the way, the whole effort took about 30 minutes and we did meet our schedule. That's the power of collecting data at a personal level and having an organized way to use it!"

[14] These figures are described in Appendix A and in Section 4.3.

"Our schedule reliability is now +/- ten percent from –50/+200 percent and our defect density at the team level has been reduced by over 50 percent."

"One of my first projects as an embedded systems programmer finished on the day we planned to finish six months earlier. I attribute the success to planning at a better granularity and making full use of the earned value tracking. The day we got 100% earned value was the day we planned to get 100% value, and we as a team celebrated like we had won a basketball game."

"Multiple projects in our organization have been able to keep within their time schedules (+/- three weeks) over a six-month span. This is something we [had] not been able to accomplish in the past. This is one of the reasons that management is very happy with the TSP process."

"Our plans are much more detailed and all the involved developers understand them. As a consequence, we deliver what we planned, on time."

"Our most important tool is the weekly meeting. We make sure to make the meeting data-centered. ENGINEERS LOVE EARNING VALUE! This is important; we ensure that an engineer's plan allows them to earn value every week"

"Immediately after our launch, the team leader had a death in the family and needed to fly home. He was gone for two weeks. As a result of the TSP, the whole team fully understood what needed to be done on the project, and the team never missed a beat. Without the TSP launch, the team probably would have only been half as productive."

"…[TSP is a] transparent project management paradigm—everybody has a common understanding of the plan and everyone knows what is going on in the project and where we are in the project at any time."

"I liked the level of detail that went into [the] initial plan, and the constant awareness of the schedule. [It] allowed us to make adjustments as the project went on, instead of waiting for a major milestone."

"Measuring progress helps generate progress."

"It provides better focus for the software developer on tasks to be done."

"At our post-mortem, our management made the following comments about the TSP: 'Realistic plans based on rigorous statistical analyses,' 'Sustained pace throughout the project,' and 'Increased management confidence in project estimates.'"

6.2.4 Commitment

What if someone built a detailed plan and no one followed it? People are often surprised that TSP launch meetings are so tumultuous. But when people are getting ready to make a commitment, they put a lot of energy into discussing and resolving issues. By the end of the launch, the team has developed a detailed plan, and not only is each team member committed to the plan, but each team member knows every other team member is also committed to the plan. This helps build team cohesion and a commitment that sustains the team through difficult times. Each team member leaves the launch with a portion of the team plan that he or she owns. This individual plan does not reside on the team leader's computer, or on a conference room wall, it belongs to the individual. The individual manages it, uses it, and changes it, several times a day if necessary. The team then meets weekly, the individual plans are consolidated into a team plan, and the team manages the team plan together. Figure 20 summarizes data from launch evaluation forms that ask about commitment to the plan.

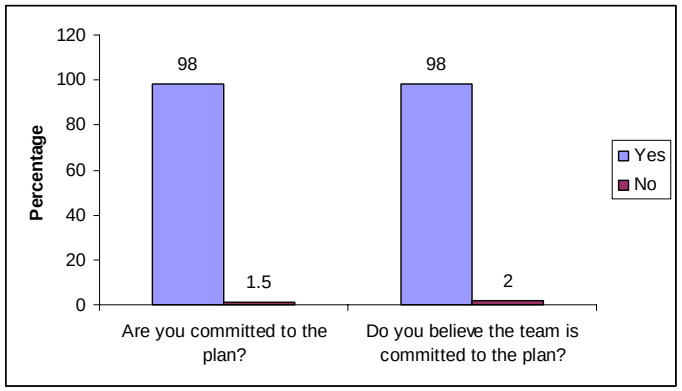


Figure 20: Commitment to the Plan

"I was launching a team. We had just started meeting 8, and had started to put together the presentation of our plan to management. The team was primarily composed of young engineers, with the exception of Lisa, who had quite a bit of experience at the organization. She had been skeptical of the TSP and had not fully participated in the launch meetings up to that point. As we got into meeting 8, I asked each engineer if they believed in the plan. Each engineer said yes, until I got to Lisa, and she said no. Lisa felt that many of our estimates were too conservative. So we went back and spent 45 minutes re-examining the plan, but this time, with the benefit of Lisa's knowledge and participation. I think for the first time Lisa understood what the TSP was all about. Up to that point Lisa felt that once we made an assumption and documented it in the plan, the plan could not be changed. Lisa now understood that the plan was not rigid, that we could change the plan whenever we learned more about what we were doing, and that this was actually encouraged. The team learned an invaluable lesson about the importance of having each and every team member committed to the plan. This team ended up meeting their schedule, and Lisa is now one of the strongest proponents of TSP in the organization."

"During one of my first team launches, the team was discussing the risks to their plan. I advised the team not to discuss risks of ordinary living, such as if someone gets hit by a bus. We were about to move on, when one team member said, 'I suppose I should mention that I've accepted a job with another company.' At first I was taken aback, but then I realized that the team was making a real commitment to each other and this engineer could not make that same commitment and wasn't comfortable pretending to do so. In fact, this engineer later told me that the launch made him disclose the fact that he was leaving earlier than he would have done otherwise, because he could not let the team go ahead and plan as if he would be there to help."

"Do NOT take over during the launches. Coach and guide, but don't do anything FOR the team. IF THEY DON'T CREATE THE PLAN THEMSELVES, THEY WON'T OWN IT AND IT WON'T BE IMPLEMENTED!! Take my word for this, I am WELL-versed in how to do it incorrectly!"

"The first time, I worried that the team would not jell. But as the launch progressed, the team jelled, and I learned not to be nervous, and let them go."

"By the third day of the launch, it was clear that the team leader was not fully engaged with the team. The team was so concerned that they discussed their concerns with their launch coach. The launch coach then had a heart-to-heart conversation with the team leader. It turns out that the team leader was having personal problems and had issues with the complexity of the project. The team leader felt that he would not be able to adequately lead this effort and stepped down from his position. The team was surprised, but they supported their team leader's decision. The team completed their launch and proposed one member as team leader during the management meeting. The team jelled more than ever and the old team leader is walking around with a smile on his face for the first time in months."

"This really feels like a tight team."

"I feel included and empowered."

"It forces team coordination to talk about and solve problems—there's no pigeonholing."

6.2.5 Minimum Schedules

TSP teams are not satisfied with just meeting schedule commitments. They want to create the best products in the shortest period of time. TSP teams follow several strategies in order to get minimum schedules.

Balanced Workloads

The first strategy discussed here is to balance workload among team members, so that no team member becomes the "long pole" in the schedule and prolongs the entire project. Figure 21 shows how schedules can be minimized when team members allocate tasks evenly among themselves. The team was able to shorten the project schedule from over 40 weeks to just over 20 weeks by re-allocating some of Engineer I's and Engineer B's work.

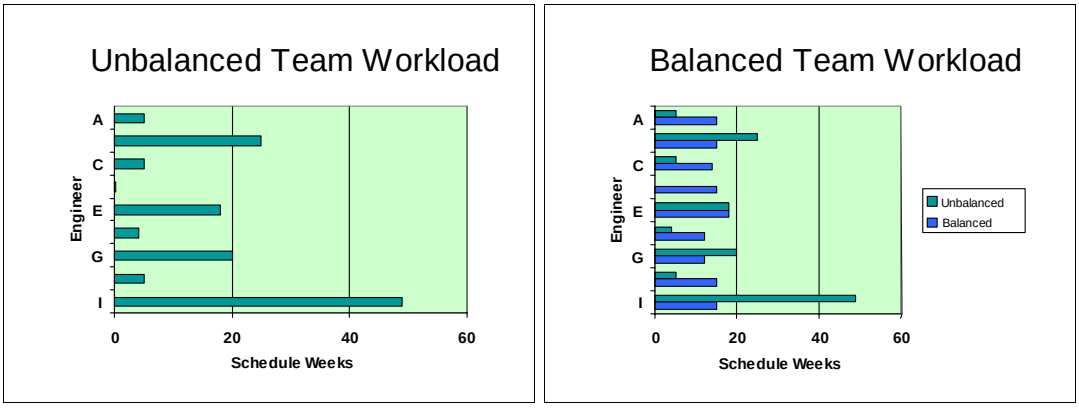


Figure 21: Balanced Team Workload

"During meeting 6, the team assigned resources based on their typical specialties. This resulted in personal plans that were completely unbalanced. The person with the earliest plan was to finish at 6 weeks and the person with the longest plan was to finish at 50 weeks, with everyone else spread in between. The team leader was worried about how he would fix that. I told the team leader that the team would figure it out and took him out of the room. We returned about 45 minutes later to find the room full of energy, with 15 people gathered around whiteboards trading tasks and figuring things out. The team had narrowed the imbalance to a minimum of 20 weeks and a max of 28 weeks. Our old planning methods would not have found the load imbalance until much later."

"At the beginning of the launch, the design manager was convinced that only he could do certain tasks. In meeting 6, when it became clear that the design manager was the long pole in the team schedule, the design manager finally recognized that he couldn't be the only one to do those certain tasks. He came up with an approach where he would train other team members to complete work that was currently assigned to him."

High-Quality Products into Test/Reduced Time in Test

Another strategy used by the TSP to minimize schedule is to focus on quality. Through careful quality management, TSP teams are able to reduce the number of defects entering the formal testing phases and thereby reduce the percentage of development time spent in formal testing. TSP teams find that the extra time they spend in reviews and inspections is more than made up for by the time saved during test. Figure 22 shows that because the team removed defects early in the development cycle (although not as many as planned), the time they spent in system test was minimized, thus minimizing the overall schedule.

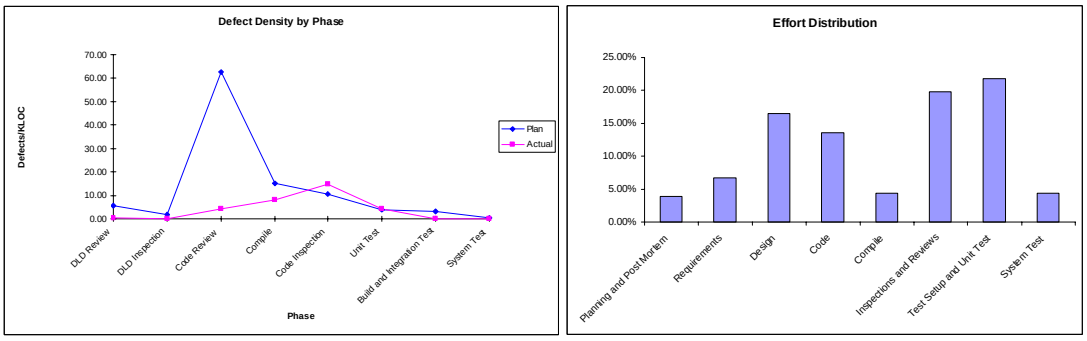


Figure 22: Defect Removal Profile and Effort Distribution

"My first TSP-based team recently finished their system test. They had three system test defects in 7400 lines of code. No defects were code- or design-related; they were either install or documentation—each of which took about five minutes to fix. System test took less than five percent of the overall project effort."

"The system test engineers became convinced that TSP was worthwhile when they realized that they were going from tracking down software bugs in the lab to just confirming functionality. Our first project: certified with ten times increase in quality with significant drop in cost to develop. Follow-on project: certified with NO software defects delivered to system test or customer."

"The first TSP team I coached was surprised when unit test was completed in half a day. They said they had done a prototype of this code before the project started and it took 1.5 weeks to get it to work well enough to see any results. They have found only two defects since the code has been integrated with the rest of the software."

"Seventy-five percent reduction in defects entering into integration testing through the use of inspections and reviews during the development process."

"Our team met both the dates and the result was defect-free in system and acceptance test. The principal software engineer on this project said, 'I would like to offer a brief endorsement of the TSP. I do NOT think that we could have made these tight deadlines without the use of the process. My estimation is that we have saved two to four calendar weeks in an eight-calendar-week project because of using the process. I think the team has been very diligent in applying the principles involved in the process, and we have reaped the benefit in that we have discovered very few integration issues.' Data shows that the team achieved 78.5 percent yield and that having to fix 190 code defects in test would have significantly extended the schedule."

"PSP really sells you on the idea about finding defects early in the process. It really does make a difference at the end. We thought it wasn't going to work. But we all became converts. In doing the work, you are producing valuable data along the way. We improved productivity…improved it greatly. I worried because I have seen too many people more interested in the process than in the product. You are finishing smaller products at more regular intervals."

"The review and inspections across layers help reduce cross-team application/interface defects and security vulnerabilities which may otherwise go untested."

"It was nice to be associated with a project that had few defects."

6.2.6 Barriers to Success

Not all TSP implementations are successful. Unfortunately, the SEI has not received data from a team that has failed. We have data from projects that were cancelled midstream or where teams were told to stop using the TSP when management sponsorship was lost. The comments in this section show that the biggest risks with the TSP are the same risks associated with technology transition in general: lack of management sponsorship, introduction costs, and resistance to change.

"When the senior manager left, we lost our sponsorship for the effort because there wasn't any buy-in from the Development Manager."

"Manage your sponsor list. Our effort has taken so long, I have only recently realized that one by one, every sponsor that had supported our TSP effort has moved on to a different job—and it happened so slowly that I didn't notice until we had a problem."

"Many managers will fail in accepted ways rather than fail trying something new."

"It's extremely frustrating to see the benefits and not be able to implement due to management issues."

"While everyone is in 'full support' of getting the software developers PSP–trained, when it comes time to send them to class, it is hard to get project relief or to get the managers to release the developers to come to class in the first place."

"We seemingly clearly show management the benefits of TSP and how the training time of three weeks[15] *is paid back within nine months. For instance, one project in our company found zero defects in system test and has yet to have a customer-reported defect after several months of usage. However, we still routinely get back the comment "we just can't afford three weeks training right now." How can you not afford it if it's a wash in nine months time and you've reduced cycle times from that point on?"*

"Because of the high up-front cost, management is reluctant to enforce its use. I think this will change shortly…at least I hope so."

"There has been immense pressure to shorten the course. Even with compelling evidence of the benefits, the training time is a seemingly insurmountable barrier. If there was some way to streamline the course so that it could be completed in two weeks, I think [that] would make a big difference."

[15] The PSP For Engineers course requires two weeks of time in class, and additional time outside class to complete some programming assignments, and mid-term and final reports.

"The length of training time is a major barrier. I'm trying to get people for a second class right now. There was no tool support that would carry PSP from the classroom to on-the-job use. There were some negative feelings from the first and second classes that had led to a lot of grass roots non-enthusiasm. Part of this negative feeling is due to class material. Part was due to lack of tools. Part was due to poor teaching. Part was due to a poor selection of students."

"Unable to gain initial commitment—very hard to get people interested in actually committing to try it due mainly to the perceived disruption in ongoing projects for the required training."

"Initial commitment is extremely difficult. Lack of management support is primarily based on the up-front investment (cost) and time they have to go without their tech staff on line."

"Time investment is accepted but logistics of timing course offerings is a challenge."

6.2.7 Problems You Had With the TSP

As with any technology, there are always opportunities for improvement, and the TSP is no exception. Here are the comments we received about the shortfalls of the TSP.

"TSP role managers were revised several times without being able to have them function properly."

"Management needs to be able to measure TSP teams against other projects. Since the other project are not measuring and reporting anything, management doesn't understand how well the TSP teams have done. I don't know the solution to this problem."

"Earned Value is a great way to plan and track development tasks, but is not as useful for schedule-driven phases of a project. When launching a new project phase, look critically at the types of reporting that will go on, and guide the team toward more or less dependence on EV vs. PERT or Gantt. Never ignore either earned value or schedule/dependencies, but simply shift the emphasis depending on the needs of the project.

"No tool support. SEI's TSP tool is not sufficient at all."

"Some aspects are good (reviews and inspections. I am not yet sold on it."

"Easy-to-use and low-cost tools are important to get PSP off the ground. While the TSP prototype tool is usable, SEI really needs to get the requirements document out to the community and embrace the development of tools. Without good tools, it will be difficult to move PSP/TSP forward."

"Currently, the most common issue is the TSP spreadsheet. The developers are finding that small mistakes in data entry can cause a lot of grief (shared by coaches). Getting a more robust tool would definitely help."

"Problems with using the TSP tool discourage [the] user from recording defects faithfully."

"Tool issues are preventing us from properly consolidating individual, team, and organization data. Multiple data entry is frustrating to practitioners."

6.2.8 General Comments

Not all anecdotes can be classified. In this section, we have included general comments we have received about the TSP. One thing that is obvious from these comments is that TSP teams love data.

"The TSP has given the engineers in my organization a common vocabulary. I first saw it in the PSP for Engineers course. Exercise 4 in the course is the 'turning point.' Up until then, most students are operating by rote with very little understanding. This is to be expected, since most engineers have never had any exposure to process and they therefore do not have a meaningful vocabulary with which to express process concepts. By lecture/exercise 4 they have learned enough of a process vocabulary to be able to express themselves. They can also begin to discuss the issues with each other—and they do. I find it very rewarding to watch this transformation take place."

"The topic of on-task hours was a point of major discussion, both within the team and in meeting 9. I showed the team pages 58-61 of Winning with Software[16] *and they photocopied these as part of the meeting 9 handout. The team was pleasantly surprised to find in meeting 9 that their failure to meet the original deadline was not a cause of management anger; rather, it led to a fruitful discussion of what was possible."*

"We had almost 100% increases in productivity."

"Our organization's first PSP-based project had 20 percent improvement in productivity compared to historical average, one of the lowest delivered defect densities ever in this organization, and the best schedule performance ever in this organization."

"One project increased its delivered quality by 10 times and reduced its effort by 20 percent compared to a previous project."

"So far, the experience has been a very positive one. The developers are finding it a much better way to run projects. Management is seeing the benefits from much improved schedule planning. The groups that have managed to record their quality data also seem to have produced quality code. We are still struggling with some groups to get their quality data. It appears to be a worry that management will somehow view them as writing defective code. We have turned the argument around by saying that if they do not record the defect data, then we will assume that the bugs were shipped (since the number caught is far less than their quality plan suggested). This has helped in a couple of the projects. Also, having management review the functionality at the end of each cycle has helped people focus on the job at hand."

[16] [Humphrey 02]

"Much better alignment of the team to management and customer goals. (Unfortunately, I am unable to quantify.)"

"First project: four times defect reduction in integration and test, doubled productivity, and shipped on time. Next project: 10 times reduction in integration and test, doubled productivity, and doubled functionality at the same time. Another project: 5 times defect reduction in test, shipped on time. Another project: 30 percent increase in productivity in six months. Another project: doubled task hours in one year."

"During a postmortem, an individual stated that the IRTL [Issue and Risk Tracking Log] was a total waste of time because none of the risks came true. I reminded him that we spent considerable time during each weekly meeting ensuring that all were actively being worked. Since no risks came true, the team should consider the IRTL review to be a complete success! The response was 'O yeah!'"

"You actually get your money back after 1200 LOC."

"Quality was better by a factor of two. Estimates were very good. Team spirit, cooperation, dedication, and collaboration; risk management was effective, the launches worked, reviews and inspections improved quality, daily meeting, team visibility, and happy with integration test effort."

"Expanded responsibilities of each team member beyond 'just their development work,' has resulted in better exchange of ideas among the team members."

"A comment from one team member towards the end of day 3 (meeting 6): 'I feel that TSP has something for me as a developer.' I think he meant that he originally felt that TSP was merely a management tool."

"The first TSP pilot in our organization shows that productivity has increased by 17.5 percent when comparing with non-PSP engineers. We had less data to come to a conclusion regarding quality. So we need another pilot."

"Regardless of how much complaining is done over relaunching, we continue to use it because time on task has improved by twice over and defects in test have been reduced by at least six times."

"I won't run a project any other way."

"A more disciplined process allowed me to do a better job, and allowed me to balance my job with other aspects of my life."

“I am a very creative person. I liken doing software to an artist painting a picture, and so I still worry about the PSP structure taking some of the fun and creativity out of the software process. PSP tends to distill the repetitive measurable tasks out of the creative and innovative ones that occur early in the design phase. The purpose of design is to provide an early analysis that leads to products with fewer of the more costly defects later. You have to have a good design to get good code.”

“We had two very successful TSP pilots and then we lost our TSP sponsor. In the first pilot, it took us a little over half a day to test each 1000 lines of code. In the second pilot, it took us a little under half a day to test each 1000 lines of code. In our third project, without the TSP, we have already spent over seven days to test each 1000 lines of code, we are still finding defects, and have not finished testing yet.”

“This is the hardest, most enjoyable, personally rewarding thing I have done outside of growing a family.”

6.3 Anecdotes – Conclusion

The anecdotes presented in this section show how the TSP introduction strategy builds skills and prepares a team for using the TSP, how the launch creates a cohesive team that is committed to the team goals, how the team develops a plan with the minimal schedules, and how teams focus on quality throughout the project life cycle. Some problems faced by TSP teams are also described. The anecdotes describe how people internalize their experiences with the PSP and the TSP.

7 Conclusion

We wrote this report to provide updated results on the use of the PSP and the TSP. We started by describing the experiences of a first-time team to illustrate how the TSP creates an environment where skilled engineers can apply disciplined methods to achieve schedule and quality goals. This team was able to achieve significant improvements on their first use of the TSP. These impressive results were not atypical, as seen by data summarized in Section 5. The individual perspective provided in Section 6 illustrates success from a personal point of view. People like doing excellent work and the TSP enables them to do so.

At the surface, the PSP and the TSP seem exclusively like planning-driven, data-oriented technologies. However, it is also the human interactions enabled by the TSP that allow individuals and teams to be successful. You see this same conflict in the balance between disciplined and creative work. People feel that discipline prevents them from doing creative work, when in fact the opposite is true. The same holds true with teamwork and data. In order for a team to jell, they need the data to manage their routine tasks. In earlier sections of this report, we presented figure after figure of project data. One would be mistaken to believe that it was this plethora of data that was the sole factor in team success. Besides data, a major contributor to the success of all these teams was the commitment and ownership generated during the launch and sustained throughout the life of the project. It is the synergy that is created when a team has a common goal and each and every person on that team understands how his or her work and everyone else's work contributes to the achievement of that goal. But what does synergy really mean? Synergy is when you are struggling and your team is there to support you. Synergy is the recognition from your team when you succeed. Synergy is the pride you feel when a team member shines or the satisfaction that comes from helping a teammate. We hope that we have presented both these aspects of the TSP in this report: the measurable results produced and the non-quantifiable results experienced.

Appendix A: PSP Empirical Study Replication

This appendix shows the preliminary findings from a replication of the Hayes and Over SEI technical report investigating the impact of PSP on various aspects of individual engineer performance [Hayes 97]. We did not attempt to reproduce the depth or breadth of the original SEI technical report on this subject. Our goal was to reexamine the major hypotheses of the earlier report using the same methodology as the original work. The most notable difference between this and the original analysis is sample size: 1300 versus 300.

Generally, there were few deviations from the findings in the original Hayes and Over technical report. There are three exceptions. First, the original report found no statistical difference between PSP levels 1 and 2 in size estimation accuracy. This difference is now significant. Likewise, the original report found no statistical difference between PSP levels 1 and 2 in effort estimation accuracy. This difference is now significant. Lastly, the original report showed no statistical difference between PSP levels 0 and 1 and levels 0 and 2 of individual changes in productivity. Both of these differences are now significant. Other than these three exceptions, the analysis is in agreement with the original Hayes and Over results.

Introduction

Repeated measures Analysis Of Variance (ANOVA) was carried out on the same measures originally investigated by Hayes and Over. These measures are believed to quantify many of the important principles underlying the PSP. Further, they are representative of the substantive changes in an individual engineer's performance brought about by PSP training. Like the original technical report, the analyses below are conducted without regard to violations of the assumptions underlying the repeated measures ANOVA model. In all cases, confirmatory post hoc analysis was conducted to determine both the existence and extent of any violations. Where violations of the ANOVA assumptions were found, all measures, with the exception of pre-compile yield (as was the case in the original study) were transformed to better conform to model assumptions. In all cases, transformed variables retained their significance, thus confirming the original findings. In the case of yield, a non-parametric test, the Jonckheere-Terpstra test for ordered differences among classes, was conducted (see page 75).

The remainder of this appendix is organized as follows. First, we present a visual "tour" of the existing data. The available figure for each measure data is presented by assignment number. In the sections that remain, following Hayes and Over, we present the primary re-

sults grouped by the measures (metrics) of interest. Each section begins by restating the original hypothesis, followed by brief descriptions of the measure's selection criteria, the measure itself, discussion of group trend, and results of the ANOVA and contrast analysis.

Descriptive Data

Assignment	1	2	3	4	5	6	7	8	9	10
Effort	1312	1296	1280	1258	1240	1202	1131	1076	989	886
Size	1250	1297	1281	1259	1240	1202	1131	1075	989	886
Defects	1305	1282	1268	1229	1218	1190	1103	1027	977	870

Table 13: Number of Engineers Reporting Totals by Assignment Number

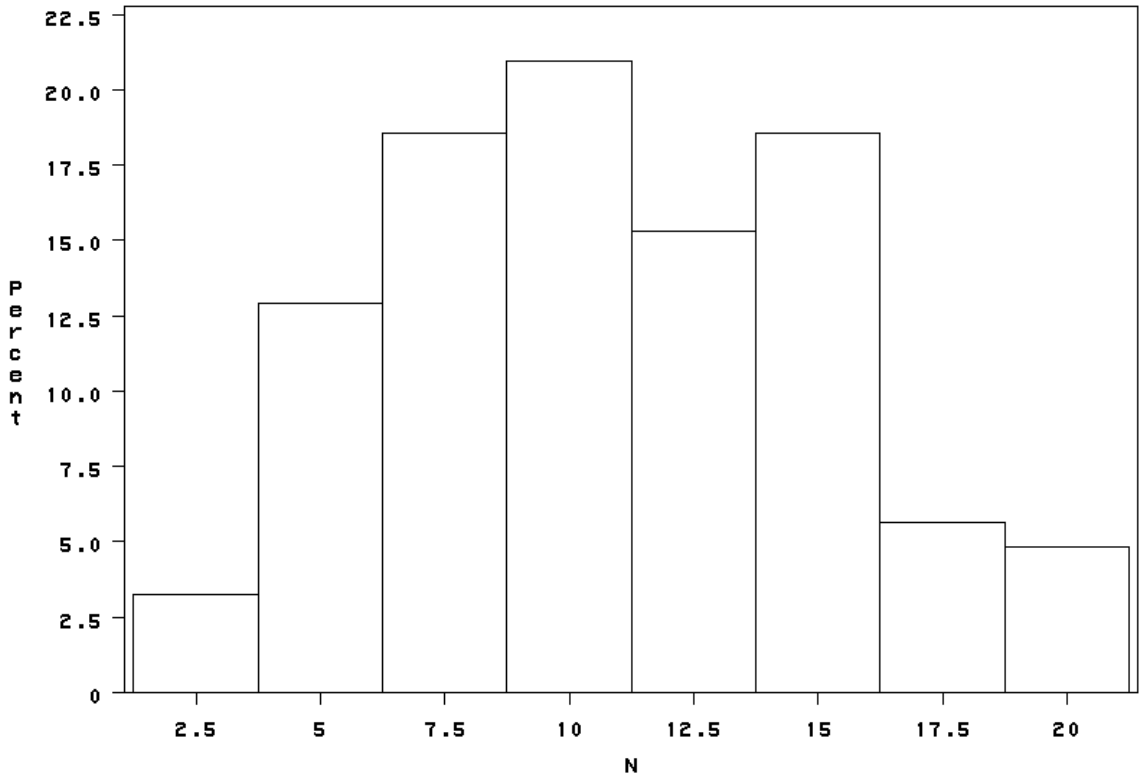


Figure 23: Distribution of Class Size

Assignment	1	2	3	4	5	6	7	8	9	10
Planning	1295	1289	1274	1256	1239	1202	1131	1075	988	881
Design	1285	1274	1267	1221	1219	1178	1122	1070	986	884
Design Review	5	5	9	5	25	30	1057	1043	963	875
Code	1311	1297	1281	1259	1240	1202	1131	1076	989	886
Code Review	7	12	15	18	40	42	1124	1070	988	883
Compile	1286	1267	1252	1227	1214	1169	1069	1005	950	852
Test	1310	1295	1281	1259	1238	1202	1128	1073	987	885

Table 14: Availability of Phase-Specific Effort by Assignment

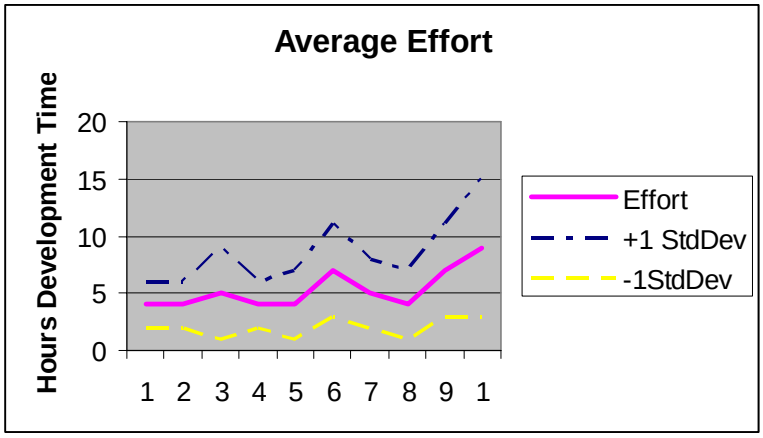


Figure 24: Average Effort by Assignment Number

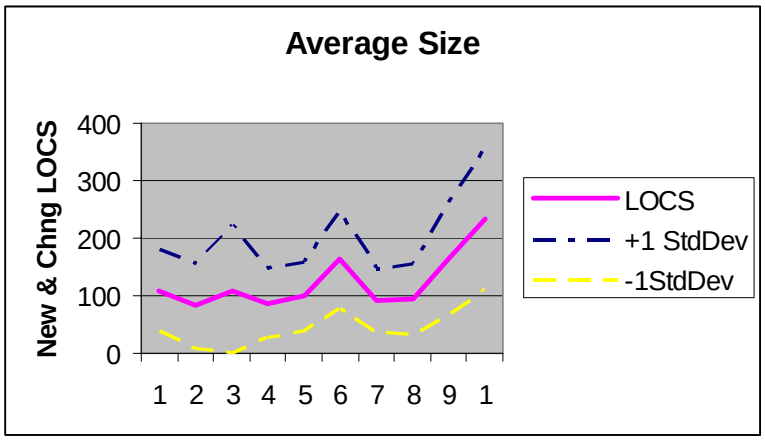


Figure 25: Average Size by Assignment Number

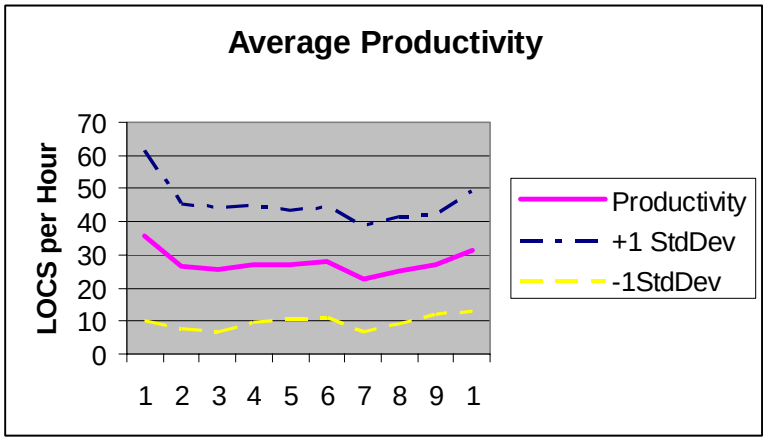


Figure 26: Average Productivity by Assignment Number

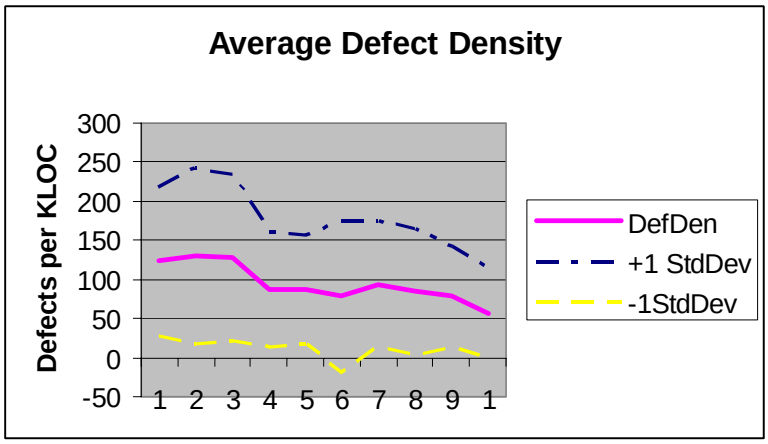


Figure 27: Average Defect Density by Assignment Number

Measure	Sample Size
Size Estimation	954
Effort Estimation	954
Defect Density	907
Pre-Compile Defect Yield	908
Productivity	964

Table 15: Sample Size for Each Measure

Size Estimation

Hypothesis:
"As engineers progress through the PSP training, their size estimates gradually grow closer to the actual size of the program at the end. More specifically, with the introduction of a formal estimation technique for size in PSP level 1, there is a notable improvement in the accuracy of engineers' size estimates" [Hayes 97].

Selection Criteria[17]

Sample Size: N = 954

Common Selection Criteria

Rather than using an average estimate and actual LOC of three assignments within a PSP level, the estimates and actuals are pooled by PSP level. This pooling serves to reduce the magnitude of outliers and provides a data point tied to performance within a PSP level. Even though data is pooled across PSP levels, only data from subjects submitting complete data are retained (that is, data that can be used to compute each measure for all nine programming assignments), thereby creating a completely balanced design as required by the statistical model. This pooling technique is common to all remaining measures in this study and is henceforth referred to simply as "pooling." Because PSP level 3 contains only one programming assignment, assignment 10, it was excluded from all remaining analyses.

Size Estimation Selection Criteria

To compute the size estimation measure for each PSP level, an estimation accuracy value is computed by summing the estimated LOC across the three assignments, summing the actual LOC across the three assignments, and then computing the measure as specified below. Since size estimates are not required for the first programming assignment, the pooled PSP level 0 data includes assignments 2 and 3 only.

Measure of Interest

Nominal

$$\frac{EstimatedSize - ActualSize}{EstimatedSize}$$

Transformed

$$\frac{EstimatedSize - ActualSize}{ArgMax(EstimatedSize, ActualSize)}$$

[17] All selection criteria in this appendix are adapted from Hayes and Over [Hayes 97].

Group Trend

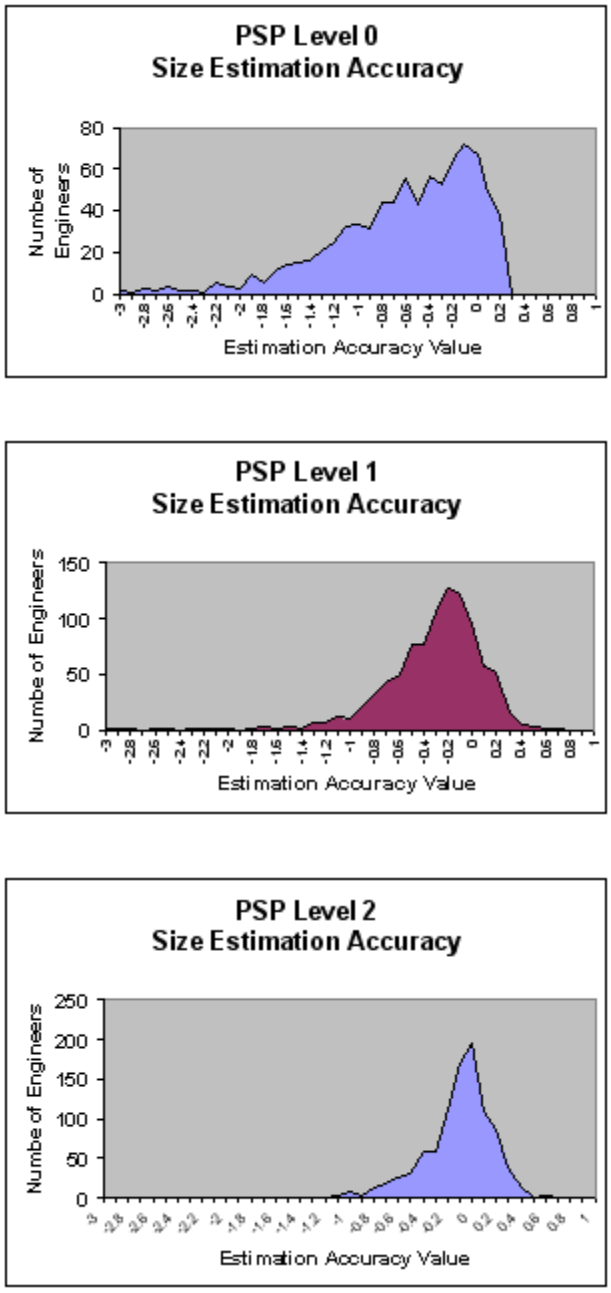


Figure 28: Group Trends in Size Estimating Accuracy

ANOVA Analysis and Findings

Analysis comparing the pooled size estimation accuracy values for each PSP level showed a highly significant difference across all three levels of the PSP ($p < .0001$). Contrast analysis also revealed statistically significant differences between adjacent levels of the PSP. That is between levels 0 and 1 ($p < .0001$) and levels 1 and 2 ($p < .0001$).

Effort Estimation

Hypothesis:
"As engineers progress through the PSP training, their effort estimates grow closer to the actual effort expended for the entire life cycle. More specifically, with the introduction of a statistical technique (linear regression) in PSP level 1, there is a notable improvement in the accuracy of engineers' effort estimates" [Hayes 97].

Selection Criteria

Sample Size: N = 954

Data is pooled as outlined above. To compute the effort estimation measure for each PSP level, an estimation accuracy value is computed by summing the estimated minutes across the three assignments, summing the actual minutes across the three assignments, and then computing the measure as specified below. Since size estimates are not required for the first programming assignment, the pooled PSP level 0 data includes assignments 2 and 3 only.

Measure of Interest

Nominal

$$\frac{EstimatedMin - ActualMin}{EstimatedMin}$$

Transformed

$$\frac{EstimatedMin - ActualMin}{ArgMax(EstimatedMin, AcutalMin)}$$

Group Trend

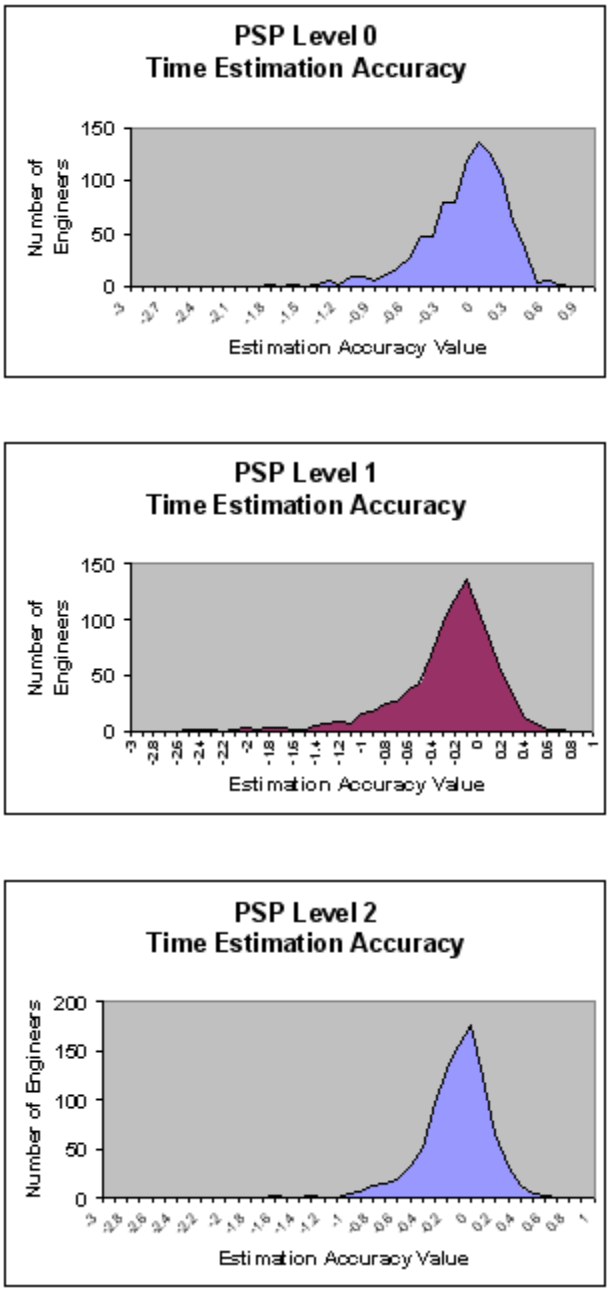


Figure 29: Group Trend in Time Estimating Accuracy

ANOVA Analysis and Findings

Analysis comparing the pooled effort estimation accuracy values for each PSP level showed a highly significant difference across all three levels of the PSP ($p < .0001$). Contrast analysis also revealed statistically significant differences between adjacent levels of the PSP. That is between levels 0 and 1 ($p < .0001$) and levels 1 and 2 ($p < .0001$).

Defect Density

Hypothesis:
"As engineers progress through PSP training, the number of defects injected and therefore removed per thousand lines of code (KLOC) decreases. With the introduction of design and code reviews in PSP level 2, the defect densities of programs entering the compile and test phases decrease significantly" [Hayes 97].

Selection Criteria

Sample Size: N = 907

Data is pooled as outlined above. To be included in the analysis, total defect removal counts, as well as defect removal counts for the compile and test phases, had to be available. In addition, actual program size (the denominator) also had to be available. An observation is included as long as the total defects injected and removed differ by no more than 2.

Measure of Interest

Nominal

$$\frac{TotalDefects}{TotalNew\ \&\ Changed\ LOC\ /1000}$$

Transformed

$$Log\left(\frac{TotalDefects}{TotalNew\ \&\ Changed\ LOC\ /1000}\right)$$

Group Trend

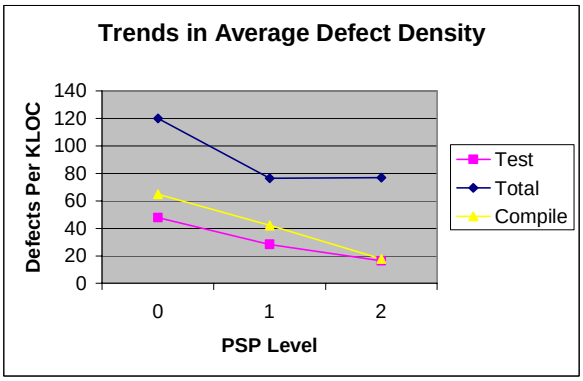


Figure 30: Group Trends in Average Defect Density

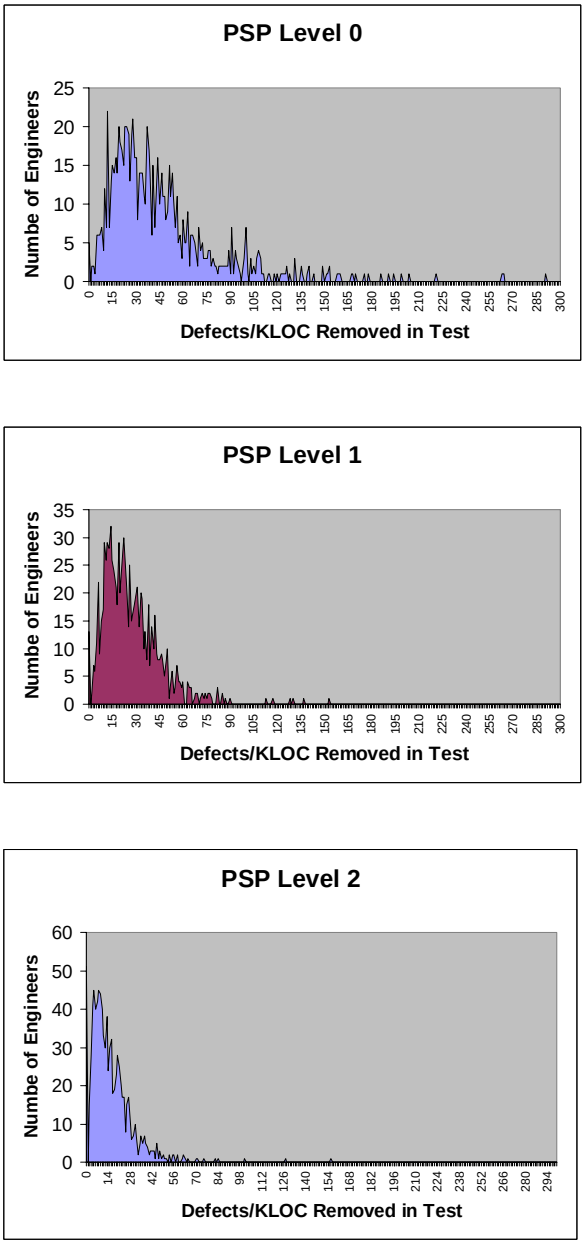
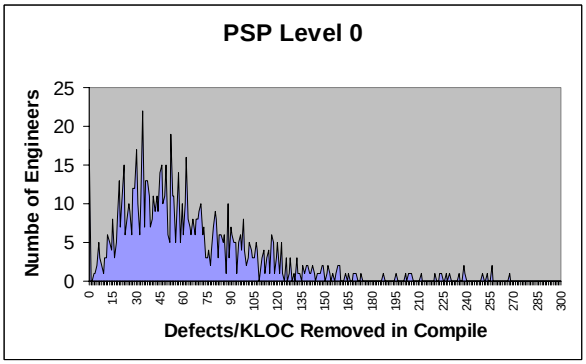
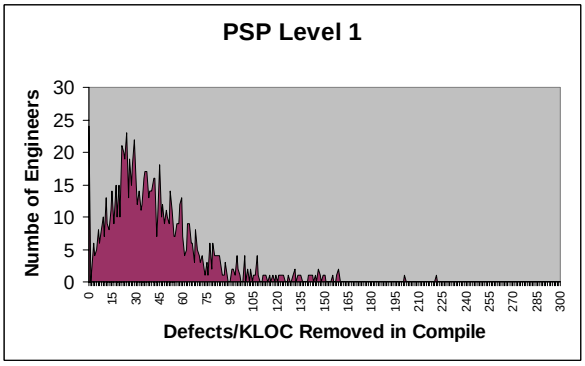


Figure 31: Group Trends in Average Defect Density





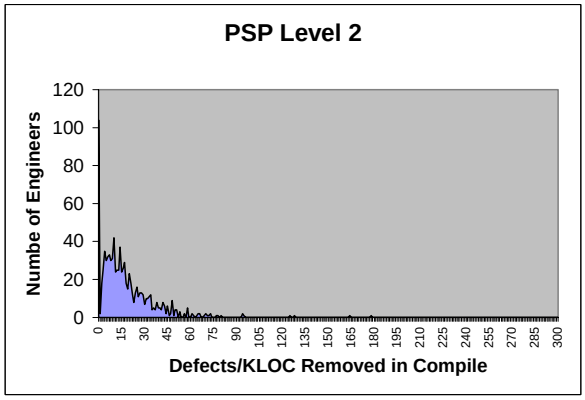


Figure 31: Group Trends in Average Defect Density, cont.

ANOVA Analysis and Findings

Analysis comparing the pooled total defect density values for each PSP level showed a highly significant difference across all three levels of the PSP ($p < .0001$). Contrast analysis revealed statistically significant differences between PSP levels 1 and 2 ($p < .0001$) but not between PSP levels 0 and 1 ($p=.27$).

Analysis comparing the pooled defect density values for the test phase for each PSP level showed a highly significant difference across all three levels of the PSP ($p < .0001$). Contrast

analysis also revealed statistically significant differences between adjacent levels of the PSP. That is between levels 0 and 1 ($p < .0001$) and levels 1 and 2 ($p < .0001$).

Analysis comparing the pooled defect density values for the compile phase for each PSP level showed a highly significant difference across all three levels of the PSP ($p < .0001$). Contrast analysis also revealed statistically significant differences between adjacent levels of the PSP. That is between levels 0 and 1 ($p < .0001$) and levels 1 and 2 ($p < .0001$).

Pre-Compile Defect Yield

Hypothesis:
"As engineers progress through the PSP training, their yield increases significantly. More specifically, the introduction of design review and code review following PSP level 1 has a significant impact on the value of engineers' yield" [Hayes 97].

Selection Criteria

Sample Size: N = 908

Data is pooled as outlined above. To be included in the analysis, defect injection and removal counts for the pre-compile phases had to be available for all nine assignments.

Measure of Interest

Nominal

$$\frac{precompile\ defects\ removed}{precompile\ defects\ injected}$$

Group Trend

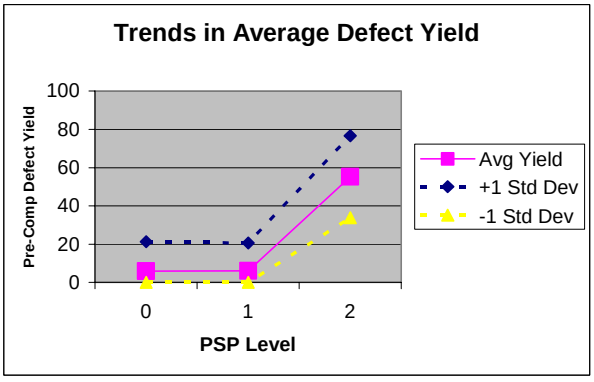


Figure 32: Group Trends in Average Defect Yield

Assignment	N Obs	Lower Quartile	Upper Quartile	Mean	Maximum	Median	Std Dev
1	1312	0.00	0.00	5.20	100.00	0.00	15.78
2	1296	0.00	0.00	5.27	100.00	0.00	16.16
3	1280	0.00	0.00	5.16	100.00	0.00	14.22
4	1258	0.00	0.00	5.76	100.00	0.00	16.63
5	1240	0.00	0.00	5.82	100.00	0.00	16.66
6	1202	0.00	0.00	6.03	100.00	0.00	15.21
7	1131	33.33	75.00	52.80	100.00	56.25	29.75
8	1076	33.33	80.00	54.89	100.00	57.52	31.91
9	989	40.00	71.43	53.49	100.00	55.56	26.16

Table 16: Yields for Each Assignment

ANOVA Analysis and Findings

Analysis comparing the pooled pre-compile defect yield for each PSP level showed a highly significant difference across all three levels of the PSP ($p < .0001$). Contrast analysis revealed statistically significant differences between PSP levels 1 and 2 ($p < .0001$) but not between PSP levels 0 and 1 ($p=.82$). Lack of variation in yield between the assignments can be seen both in Figure 32 and Table 16. Engineers are very poor at removing defects early in the development phases for the initial assignments that compose PSP levels 0 and 1. The introduction of formal code and design reviews after PSP level 1 seems to dramatically improve yield performance.

As with the original analysis, yield presents a particularly thorny problem with respect to violation of the multivariate normality assumptions underlying the repeated measures ANOVA multivariate test statistics. Specifically, PSP levels 0 and 1 are grossly non-normal, while PSP level 2 is fairly well-behaved. Therefore, there is no single transformation that can transform all the pooled data in the desired way.

Following Hayes and Over, we conducted a non-parametric analysis of the pooled yield data—a Jonckheere-Terpstra test. The Jonckheere-Terpstra tests the null hypothesis that the distribution of the response variable does not differ among classes. It is designed to detect alternatives of ordered class differences. For such ordered alternatives, the Jonckheere-Terpstra test can be preferable to tests of more general class difference alternatives, such as the Kruskal-Wallis test. In the case of yield, the Jonckheere-Terpstra test rejects the null hypothesis that the distribution of the response variable does not differ among PSP levels in favor of the alternative hypothesis of increasing order from PSP level 0 to PSP level 2 at significant alpha level ($p < .0001$).

Productivity

Hypothesis:
"As engineers progress through the PSP training, their productivity increases. That is, the number of lines of code designed, written, and tested, per hour spent increases between the first and last assignments" [Hayes 97].

Selection Criteria

Sample Size: N = 964

Data is pooled as outlined above. To be included in the analysis, new and changed lines of code and time spent completing the assignment had to be available for all nine assignments.

Measure of Interest

Nominal

$$\frac{Total\ New\ \&\ Changed\ LOCS}{Total\ Time\ Spent\ /\ 60}$$

Transformed

$$Sqrt\left(\frac{Total\ New\ \&\ Changed\ LOCS}{Total\ Time\ Spent\ /\ 60}\right)$$

Group Trend

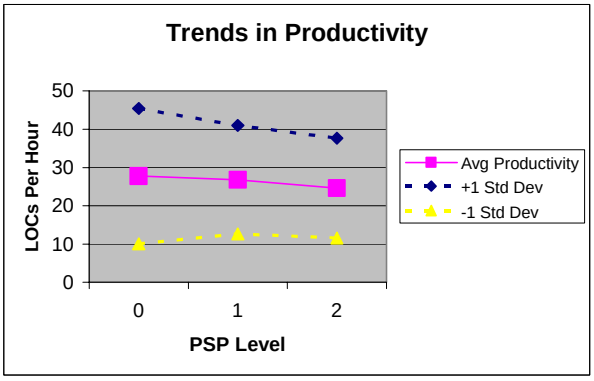


Figure 33: Group Trends in Productivity

PSP Level	Mean Value	Std Dev
0	27.7506573	17.6703262
1	26.8389425	14.1803821
2	24.6242135	13.056082

Table 17 Average Productivity

ANOVA Analysis and Findings

Analysis comparing the pooled productivity values for each PSP level showed a highly significant difference across all three levels of the PSP ($p < .0001$). Contrast analysis also revealed statistically significant differences between adjacent levels of the PSP. That is between levels 0 and 1 ($p = .01$) and levels 1 and 2 ($p < .0001$). In general, the direction or "sign" of these differences is the same as between adjacent means. In this case, productivity is shown to decrease across PSP levels, and this decrease, while small, is statistically significant.

Repeated Measures ANOVA

The primary statistical technique used to test the impact of PSP training in this report is the repeated measures ANOVA. This technique is well suited for a situation in which measurements are taken repeatedly on the same subjects. The following scenario describes one of the motivations for use of this statistical model.

> *In the study of the effect of a training course, a researcher may wish to compare scores on a pretest with scores on a test administered after training. Differences between these two test scores (given a host of other experimental conditions) are then attributed to the effectiveness of the training. In order to make a generalizable statement from such a study, individual differences must be accounted for before the two sets of measurements can be meaningfully compared. To simply state that the average post-test score for the group is greater than the average pretest score could overlook the possibility that the majority of subjects did not change at all, but one or two subjects scored significantly higher on the post-test than they did on the pretest.*

By performing the statistical test on the average change of the individuals (rather than the change in group average), the repeated measures ANOVA provides a more rigorous analysis of data collected over time from a single group of individuals. This additional level of precision, however, places more stringent requirements on the data collection and interpretation process.

Appendix B: Survey Questions

A survey of PSP instructors and TSP coaches was conducted over the Web in April 2002. The survey contained questions about experiences with teaching and implementing the PSP and the TSP. Fifty-four out of a possible 184 people responded. The results from the survey are used throughout this report, and some of the results are summarized and presented in this section.

Survey Questions

The survey asked questions about three areas: facts and figures about the TSP, the experiences of the respondents, and the respondents' organizations. The following is a summary of some of the questions on the survey:

- for each type of course, the number of courses taught, and the number of students taught by each respondent
- lessons learned for the PSP and TSP
- frequently cited story used when talking about the PSP and TSP (favorite anecdote, problematic situation)
- barriers experienced
- successes achieved
- notes on tailoring
- organization's background

PSP Usage

Figure 34 represents the responses received to the following question:

How would you describe your organization's current usage of PSP? Please select one.

- *not applicable (consulting/training only)*
- *no longer using*
- *in the planning stages*
- *pilots in progress*
- *implementing beyond pilots*

If you selected "no longer using," please briefly describe the reasons.

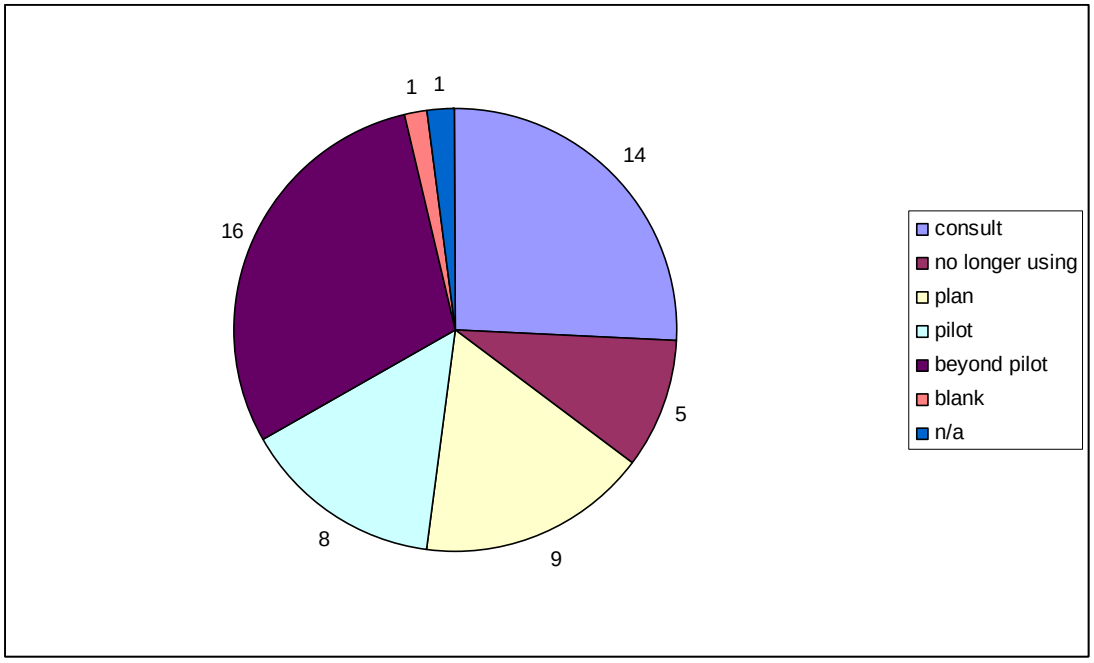


Figure 34: PSP Usage

TSP Usage

Figure 35 represents the responses received to the following question:

How would you describe your organization's current usage of TSP? Please select one.

- *not applicable (consulting/training only)*
- *no longer using*
- *in the planning stages*
- *pilots in progress*
- *implementing beyond pilots*

If you selected "no longer using," please briefly describe the reasons.

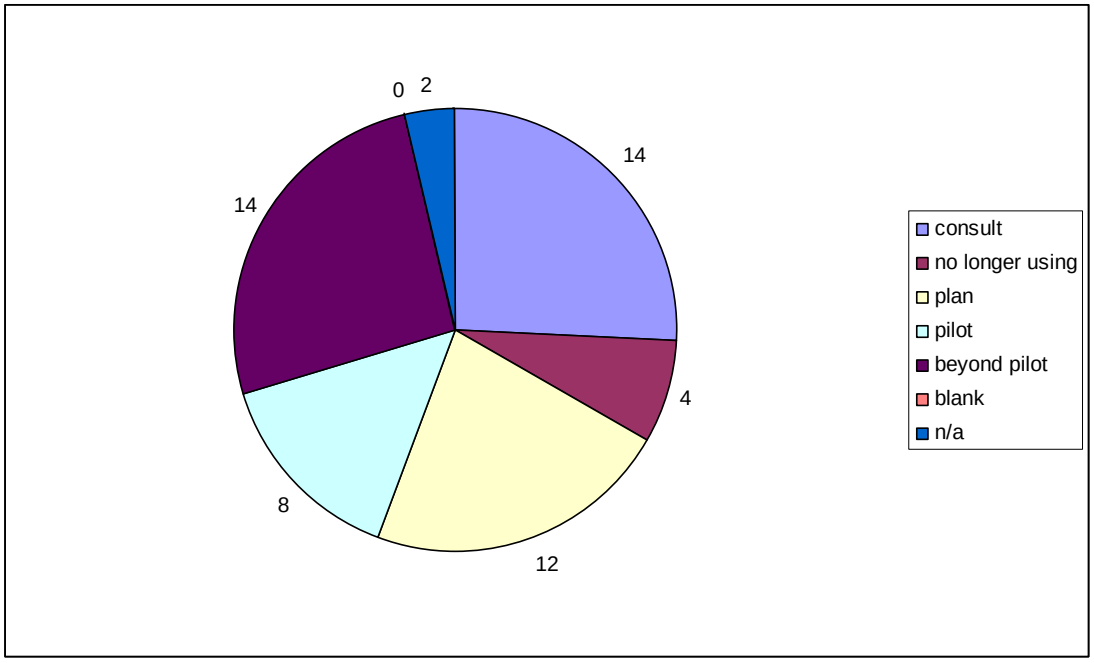


Figure 35: TSP Usage

Types of Organization

Figure 36 shows the responses received to the following question:

How is your organization best described? (Please select one)

- *Department of Defense or military*
- *Defense, military, aerospace contractor*
- *Other government agency or contractor*
- *Commercial, e.g., health, pharmaceutical, finance, insurance, telecommunications, transportation, manufacturing (please specify)*
- *Other (please specify)*

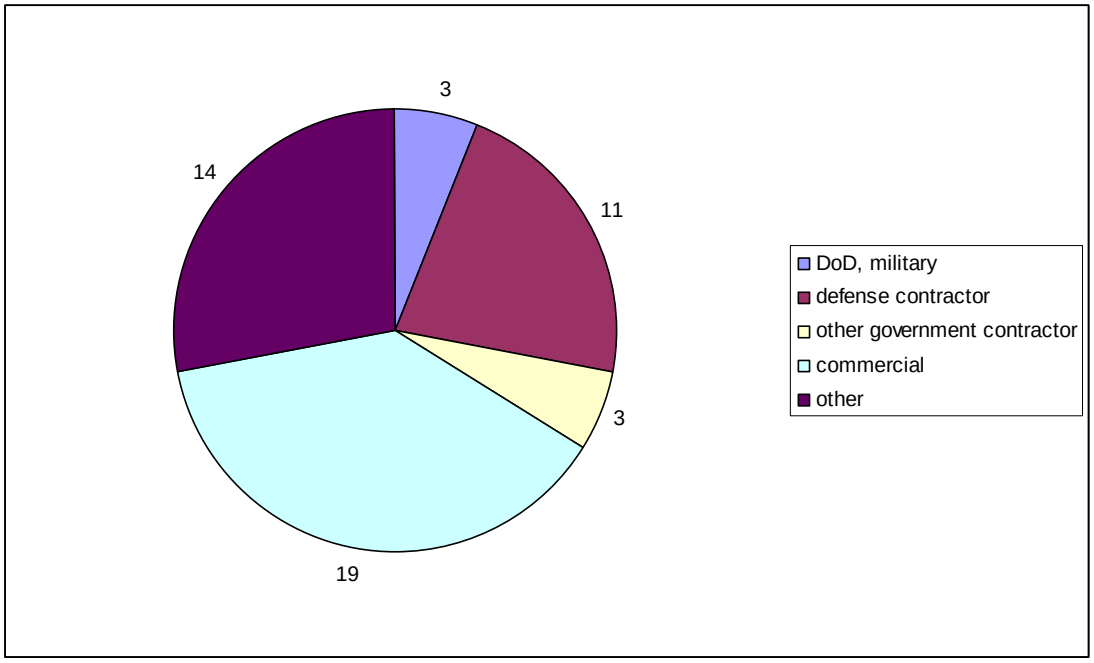


Figure 36: Organization Types

Types of Software

Figure 37 shows the responses received to the following question:

What is your organization's primary type of product?

- *Commercial shrinkwrap*
- *In-house development*
- *Maintenance*
- *Other (please specify)*

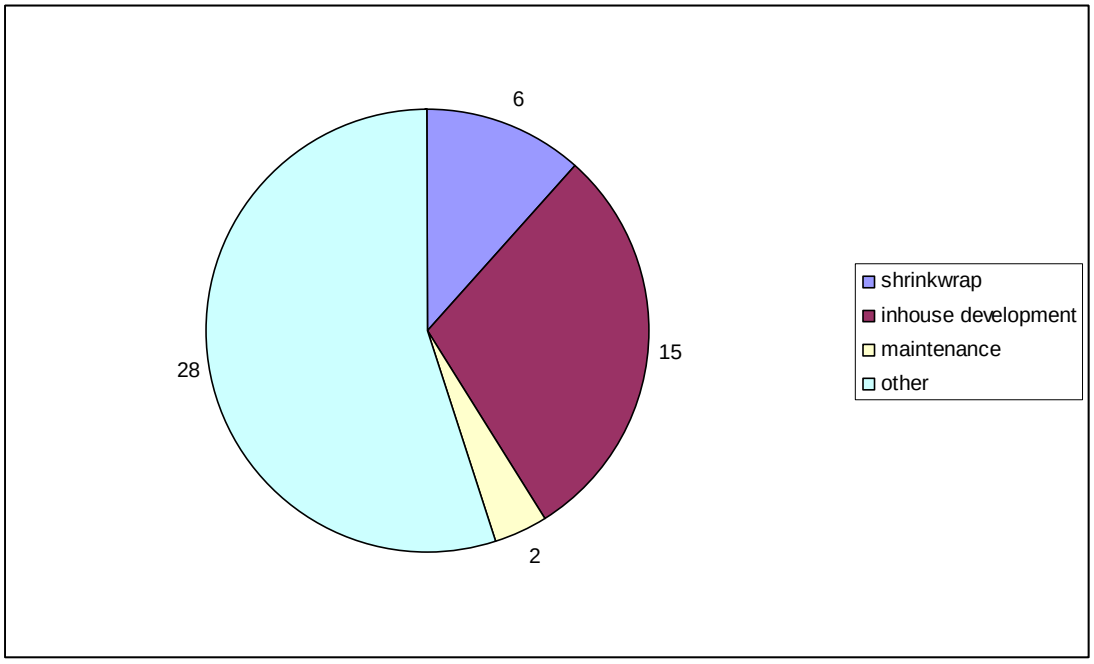


Figure 37: Software Product Types

Types of Projects

Figure 38 shows the responses received for the following question:

What is your organization's primary type of work? (Please select one)

- *Software engineering*
- *Systems engineering*
- *Both software and systems engineering*
- *Other (please specify)*

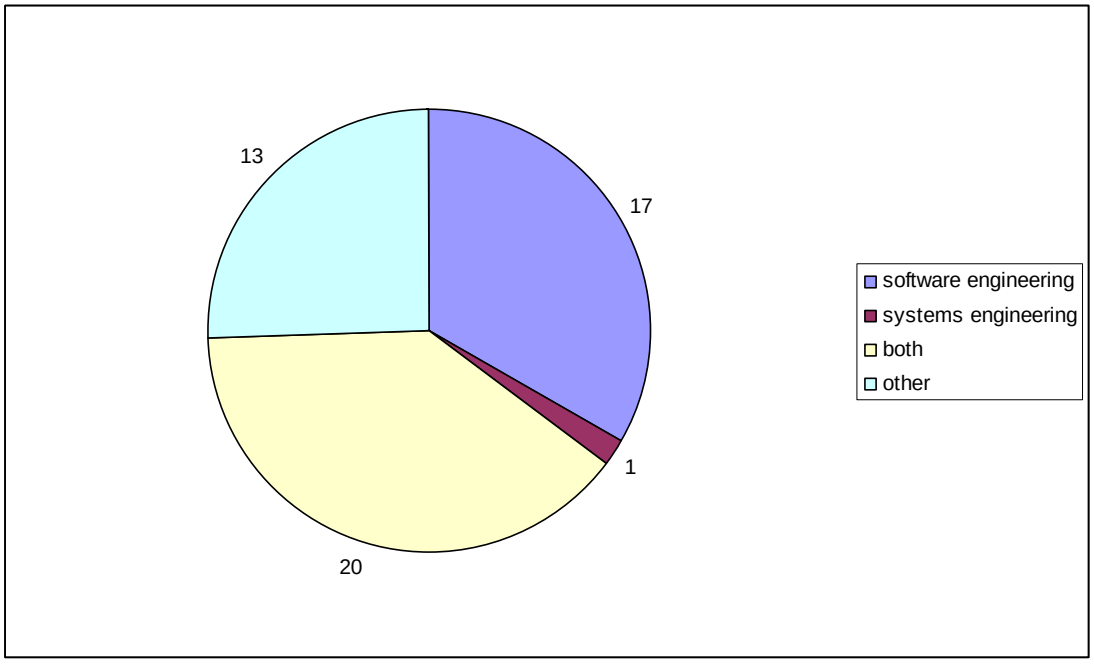


Figure 38: Project Types

References

All URLs are valid as of the publication date of this report.

[Ciurczak 02] Ciurczak, John. "Team Software Process (TSP) Experiences in the Foreign Exchange Market." *SEPG 2002* (CD-ROM). Phoenix, AZ, February 18-21, 2002. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Davis 01] Davis, Noopur; Humphrey, Watts; & McHale, Jim. "Using the TSP to Accelerate CMM-Based Software Process Improvement." *SEPG 2001: Focusing on the Delta* (CD-ROM). New Orleans, LA, March 12-15, 2001. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.

[Ferguson 99] Ferguson, P.; Leman, G; Perini, P; Renner, S.; & Seshagiri, G. *Software Process Improvement Works!* (CMU/SEI-99-TR-027, ADA371804). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. <http://www.sei.cmu.edu /publications/documents/99.reports/99tr027/99tr027abstract.html>.

[Hayes 97] Hayes, W. & Over, J. W. *The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers.* (CMU/SEI-97-TR-001, ADA335543). Pittsburgh, PA: The Software Engineering Institute, Carnegie Mellon University, 1997. <http://www.sei.cmu.edu/publications/documents/97.reports /97tr001/97tr001abstract.html>.

[Humphrey 95] Humphrey, Watts S. *A Discipline for Software Engineering.* Reading, MA: Addison-Wesley, 1995.

[Humphrey 00] Humphrey, W. *The Personal Software Process℠ (PSP℠)* (CMU/SEI-2000-TR-022, ADA387268). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tr022.html>.

[Humphrey 02] Humphrey, Watts S. *Winning with Software: An Executive Strategy.* Reading, MA: Addison-Wesley, 2002.

[Janiszewski 01] Janiszewski, Steve & Myers, Chuck. "Making Haste Deliberately." *SEPG 2001: Focusing on the Delta* (CD-ROM). New Orleans, LA, March 12-15, 2001. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.

[Jones 95a] Jones, Capers. *Patterns of Software Systems Failure and Success.* Boston, MA: International Thomson Computer Press, 1995.

[Jones 95b] Jones, Capers. *Backfiring: Converting Lines of Code to Function Points. IEEE Computer 28*, 11 (November 1995): 87-88.

[Jones 96] Jones, Capers. *Applied Software Measurement*. New York, NY: McGraw-Hill 1996.

[Jones 00] Jones, Capers. *Software Assessments, Benchmarks, and Best Practices*. Reading, MA: Addison-Wesley, 2000.

[McAndrews 00] McAndrews, D. *The Team Software Process℠: An Overview and Preliminary Results of Using Disciplined Practices* (CMU/SEI-2000-TR-015, ADA387260). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. <http://www.sei.cmu.edu/publications/documents/00.reports/00tr015.html>.

[Morgan 93] Morgan, Ben B., Jr.; Salas, Eduardo; & Glickman, Albert S. "An Analysis of Team Evolution and Maturation." *Journal of General Psychology 120*, 3: 277-291.

[Narayanan 02] Narayanan, Sridhar. "People – Process Synergy." *SEPG 2002* (CD-ROM). Phoenix, AZ, February 18-21, 2002. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Pracchia 03] Pracchia, Lisa & Hefley, Bill. "Accelerating SW-CMM Progress Using the TSP." *SEPG 2003: Assuring Stability in a Global Enterprise* (CD-ROM). Boston, MA, February 24-27, 2003. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

[Riall 02] Riall, Cary & Pavlik, Rich. "Integrating PSP, TSP, and Six Sigma." *SEPG 2002* (CD-ROM). Phoenix, AZ, February 18-21, 2002. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Schwalb 03] Schwalb, Jeff & Hodgins, Brad. "Team Software Process for Maintenance Projects." *SEPG 2003: Assuring Stability in a Global Enterprise* (CD-ROM). Boston, MA, February 24-27, 2003. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

[SEL 93] Condon, S.; Regardie, M.; Stark, M.; & Waligora, S. *Cost and Schedule Estimation Study Report* (Software Engineering Laboratory Series SEL-93-002). Greenbelt, MD: NASA Goddard Space Flight Center, 1993.

[Serrano 03] Serrano, Miguel A. & Montes de Oca, Carlos. "Using TSP in an Outsourcing Environment." *SEPG 2003: Assuring Stability in a Global Enterprise* (CD-ROM). Boston, MA, February 24-27, 2003. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.

[Sheshagiri 02] Sheshagiri, Girish. "It's Hard To Believe Unless You Do It." *SEPG 2002* (CD-ROM). Phoenix, AZ, February 18-21, 2002. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Webb 02] Webb, Dave. "Managing Risk with the Team Software Process." *SEPG 2002* (CD-ROM). Phoenix, AZ, February 18-21, 2002. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	September 2003	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
The Team Software Process℠ (TSP℠) in Practice: A Summary of Recent Results	F19628-00-C-0003

6. AUTHOR(S)

Noopur Davis, Julia Mullaney

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2003-TR-014

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2003-014

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Most software organizations are facing critical business needs for better cost and schedule management, effective quality management, and cycle-time reduction. The Team Software Process addresses these critical business needs. This report provides results and implementation data from projects and individuals that have adopted the TSP. The results show that TSP teams are delivering essentially defect-free software on schedule, while improving productivity. These data can be used for benchmarking and planning, motivation, lessons learned, and other guidance to those currently using the TSP or considering its use. The report also illustrates adoption experiences of practitioners in the field, including TSP team members, their managers, and their coaches and instructors.

14. SUBJECT TERMS	15. NUMBER OF PAGES
TSP, PSP, software process improvement, defect-free software, cycle-time reduction, software development schedule management	104

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL