14

Using the Personal Software Process

Now that you have learned the PSP, the next step is to apply it on a real development project under real schedule pressures. This turns out to be a significant challenge and it is the principal reason why the SEI developed the Team Software Process (TSP). It is also why most PSP developers work on TSP teams. The TSP shows you how to use the PSP on a development project and how you and your teammates can develop your own best practices to augment and enhance what you learned with the PSP. TSP teams also learn teamwork, how to support each other, and how to control and direct their own projects.

This chapter describes the principal challenges that software professionals face and how the PSP and TSP can help you to address them. It describes the TSP's objectives, structure, and methods, and it explains how these methods help you to be a disciplined professional software engineer. Finally, it covers the responsibilities we all face as software professionals.

14.1 Development Challenges

Although modern development work poses many technical challenges, the problems that seem to cause the most trouble are not technical at all. They concern negotiating commitments, maintaining control of our projects, delivering quality products, and

sustaining effective teamwork. To have a successful project, we must handle all of these challenges. Falling short on any of these aspects will seriously damage and possibly destroy our projects, almost regardless of how well we handle the other challenges or how fine a technical job we do in designing and building the product.

Few software development teams can simultaneously handle all four of these challenges, which is why so many software projects are troubled. The Standish Group reports that only 26% of all the software projects it has studied are considered successful. Of the remaining 74%, a full 28% either were cancelled or failed in other ways. The remaining 46% were late and seriously over budget (Standish Group 1999). For the late projects, the average delay was over 100%. Table 14.1 shows the Standish data on project success as a function of project size.

Although it is common knowledge that large software projects are almost always troubled, the key question is why. The reason is that current software practices do not scale up, and the reasons for this are related to the four challenges we face on development teams.

First, the larger the project, the less the team has to say about the project's commitments. It seems that, at least with current practices, as projects get larger, management pays less and less attention to what we developers have to say. They tell us how our projects are to be initiated, planned, and run. Though no one is picking on us, if we don't have anything to say, nobody will listen to us.

Second, project control is challenging for small software jobs; and as job scale increases, it is progressively more difficult to understand job status and to keep all the parts of the work synchronized and on schedule. Although this is a management problem, its roots are in the developers' personal practices. If we don't know where we are on a job, there is no way management can know where the overall job stands.

Third, in scaling up a process, the quality of all of the system's parts becomes progressively more important. Without high-quality parts, it is impossible to build high-quality systems. This is where the practices of the individual developers come in. Every developer is important, and with large software systems, any poor-quality

TABLE 14.1 PROJECT SUCCESS VERSUS PROJECT SIZE (STANDISH GROUP 1999)

Project Size	People	Time - Months	Success Rate
Less than $750K	6	6	55%
$750K to $1.5M	12	9	33%
$1.5M to $3M	25	12	25%
$3M to $6M	40	18	15%
$6M to $10M	+250	+24	8%
Over $10M	+500	+36	0%

part will eventually cause problems. With large or small software projects, the personal practices of every developer are critically important to the success of the overall job. This is where the PSP comes in.

Fourth, as teams grow larger, it becomes progressively more difficult to build cohesive and motivated teams and it is much more challenging to maintain team energy and commitment. This is why the TSP is so important.

Although it would be unfair to blame every software problem on the software developers, there are things we can do to ensure that our projects regularly deliver quality products on time and for their committed costs. The PSP teaches you how to do this with your personal work and the TSP shows you and your team how to do it, even with large complex projects.

Negotiating Commitments

In almost any development organization, the developers are often pressured into making schedule commitments that they think are unrealistic. The problem is that the managers are also pressured by their executives and customers to meet aggressive commitments. Unfortunately, these managers typically think that the only way to manage development work is to give their projects a sense of urgency. Unless their staff are striving to meet an aggressive challenge, managers intuitively feel that their projects will not likely be successful. Therefore, managers will push until they are convinced that you and your teammates are striving to meet an aggressive plan. The problem we face as professionals is figuring out how to get management to accept a realistic plan. With the PSP skills and the TSP process, you will be able to do this.

Although most managers are reasonable, few have a sound basis for deciding what is a good plan. They may ask for a few expert opinions, but they will primarily depend on how you react when pressed. The proper way to deal with such requests is to *always* make a plan, even if management doesn't ask for one. Say that you will do your best to meet management's goals but that you won't make a commitment without a plan. Don't say how hard the job is, that you think that a bigger team is needed, or anything else that might imply that you will not do your utmost to meet their requested date. Management has told you what is needed and you will do your best to do it, but you need a plan before you can commit to a schedule.

Only the dumbest managers want a flimsy plan that you can't meet. Managers have business commitments, and timely delivery is important. Product schedules are the foundation for revenue and expense projections. The managers may blame you for the current fiasco, but if they have too many fiascoes, they will lose their jobs. The most successful course for you and your team is to produce the best plan that you can, regardless of the pressure. Then defend it. In defending the plan, a generally successful approach is to stress the following points:

- This is the best plan we can make to meet the requirements you gave us. We can show you the plan assumptions and the historical data on which the plan is based and we can show you comparisons with similar projects. We think you will then agree that this is a tight but realistic plan.
- If you want to change the requirements or other assumptions, we will reexamine the plan to see how that will affect the schedule.
- This is a minimum-cost plan. If you are more concerned about the schedule, we could save time, but at added cost. We could also develop some schedule options if you can tell us what alternatives you would like us to consider.

You might even have alternative plans prepared in advance. The key points are to be responsive, to do a thorough job, and to make a plan before making *any* commitments. Examine any alternatives and do whatever you can to meet management's objectives, but resist any schedule or resource cuts that are not backed up with a plan.

This strategy works for individuals and for teams. With PSP training, you have learned how to make a personal plan, collect data, and use data to manage and improve your performance. On a TSP team, the TSP coach will walk you through the TSP launch process and guide you and your team in developing the plan and defending it to management. This may sound risky and counter to your organization's culture, but it isn't hard when you use the TSP. Surprisingly, based on the experiences of TSP teams to date, it is *always* successful.

Maintaining Project Control

Having a sound and detailed plan is an important first step in maintaining project control. To maintain project control, however, you must follow the plan and keep it up to date. You must also keep management informed of your progress. Keeping the plan up to date is crucial because software development is a continuously changing business and every surprise usually means more work for you and your team. Although some of this change is a natural consequence of development work, the changes that are the hardest to manage are the ones that impact requirements scope and development resources. The key to managing such change requests is to take the time to update the plan and to explain to management what the change will cost. Then you must get management's approval for the cost, schedule, resource, and product impact before agreeing to the change.

This is not a question of blame but an objective way to explain the consequences of a proposed change. You had not planned to do whatever is now requested and it will take additional time and effort. Unless management and the customer are willing to accept the cost and schedule consequences, you cannot make the change and meet the original commitment. When you and your team follow this strategy, you will be surprised at how many changes turn out to be either unnecessary or deferrable.

Following the plan and keeping management informed are two aspects of the same issue. If you don't follow the plan, you won't know where you are in the project. Then you can't inform management of your status. If you can't keep management informed of project status, they will not trust you to run the project. Then you will face all kinds of uncontrollable pressures, such as scope changes, resource cuts, or special team-member assignments. Even worse, you won't have the plan to help you negotiate these changes with management. There are many ways that management can help but they all require trust and understanding. If you know where you are and if you keep management regularly informed, they will trust you and support you in maintaining control of your project.

Delivering Quality Products

Quality management is a major focus of the PSP and TSP because without effective quality management, the inevitable quality problems will impact your project's cost and schedule, and sharply reduce the value of your products. Although you may not be too concerned with the business value of your products, you should be. Management hires and pays software developers because the value of our products is substantially more than it costs to develop them. If your projects cost too much then your products will cost too much and your customers could start looking elsewhere. If that happens too often, you could lose your job.

Sustaining Effective Teamwork

Teams are the most effective means humans have devised for doing complex creative work. Teams can be enormously effective when they work properly; without adequate guidance and support, however, they can fail miserably. Team performance depends largely on the capability of the team members to work together. To be fully effective, teams must be properly formed, guided, and led. Unfortunately, in the development world, team formation, guidance, and leadership is largely a matter of chance. There is no teambuilding process, team leaders are rarely trained in effective leadership methods, and there is little or no team coaching. As a result, software teams are rarely as effective as they could be.

14.2 The Team Software Process (TSP)

The TSP was developed to address the commitment, control, quality, and teamwork problems faced by most software development teams. It has proven to be effective at showing software teams how to capitalize on the skills of their members

and how to produce high-quality products on predictable schedules and for their committed costs. The following TSP topics are described in the next few sections:

- The logic of the TSP
- Teambuilding
- The TSP launch process
- The TSP coach
- Managing your own project
- TSP results

14.3 The Logic of the TSP

Although the TSP is not magic, it will help you to address the key issues that have continued to plague software teams. Thousands of developers have now been trained in the PSP, and many TSP teams are now consistently meeting cost and schedule targets with high-quality products (Davis and Mullaney 2003). By following a defined, planned, measured, and managed personal process, you can plan your own work, make commitments that you consistently meet, and be more productive. You will also have a more rational family life, be proud of the products that you produce, and enjoy working on a cohesive and productive team.

A basic TSP principle is that the only group that can determine the best way to run a project is the team that will do the job. This is the only group that understands the technical details of the job and can effectively map the work to the team members' skills and knowledge. After the team members figure out this "best way," the next challenge is for them to actually work that way, even when in a crisis.

The purpose of the TSP is to build and guide teams in defining their own strategies, processes, and plans, and then to guide and support them in following these strategies, processes, and plans. The PSP provides the skills that developers need in order to make personal and team plans, to track their work, and to build quality products. These skills are required to be fully effective TSP team members.

14.4 Teambuilding

A team is a group of at least two people who are working toward a common goal for which each person has a specific role and the team members must all contribute and support each other to be successful (Dyer 1984). Many studies have shown that successful teams share the following characteristics (Humphrey 2002):

1. The team members are all skilled and capable of doing their job.
2. The team has an aggressive and important goal that the members must cooperatively accomplish.
3. The team members believe that the goal is achievable, and they each have a defined role in achieving that goal.
4. The team members have a common process and plan that guides them in doing the work and in tracking their progress.
5. The team leader supports and protects the team and keeps the members informed of management issues and team progress.

These five characteristics are all essential for a group to become a high-performing team. Though the TSP launch process helps groups develop these characteristics, team motivation is crucial. Most professionals want to perform, and motivated professionals strive for superior performance. As Martin says, "Improved performance comes from motivation, from arousing and maintaining the will to work effectively, not because the individual is coerced, but because he is committed" (Martin 1993).

Team motivation is the single most important aspect of the TSP. To appreciate how the TSP addresses this need, consider what motivates you. How important is it to be working on an exciting product? Does it matter who you work with? How about the way you do the job? Do you like to be told how to work or would you rather make these decisions for yourself? Most developers want to work on important products with a great team, and do the job in the way and according to the plan that they believe is best. The TSP methods enable teams to work this way. The TSP calls these **self-directed teams**. They provide as near to an ideal software development environment as I have ever seen.

A principal purpose of TSP management training is to explain what self-directed teams are, why they are desirable, and how they should be guided and supported. This training generally convinces management to allow their TSP teams to direct their own work. They do this in return for an implicit promise that the teams will plan and manage their own work, keep management and the customer regularly informed of their progress, and produce high-quality products. Therefore, to work on a self-directed TSP team, the team members must all be trained and the team must be built. The training is done in the PSP course and the teambuilding is started with the TSP launch process. Teambuilding, however, must continue as long as the team works as a group.

14.5 The TSP Launch Process

Shortly after PSP training, TSP teams are guided through the TSP launch process by a trained and qualified TSP coach. In this launch, management and the team hold a series of ten team and management meetings during which they work through the following activities:

1. In meeting 1, management and a user representative (or the customer) meet with the team to explain why the project is needed, what management expects from this team, and what the customer or user wants. During this meeting, management outlines their goals for the team and explains why the project is important. They also describe their critical needs and any trade-off opportunities. This and meeting 9 may be attended by visitors and observers. The other eight meetings are for the team, the team leader, and the coach, with no outside observers.
2. In meeting 2, the team defines its goals and the members select their personal team roles. The goals are of two kinds: the explicit goals stated by management and the implicit goals the team members define for their own work. Although the team leader is usually selected by management, the TSP suggests eight team-member roles to provide oversight for planning, process, support, quality, requirements, design, implementation, and testing. Teams may add or delete roles depending on their project's needs. By agreeing on the team's goals and selecting their personal roles, the team members start the TSP teambuilding process.
3. During meeting 3, the team produces its initial product concept, decides on a development strategy, and defines the process needed to support the goals and strategy. A key part of building an effective team is having the team develop a common understanding of the right way to do the job.
4. In meeting 4, the team produces an overall plan for the job. In this step, the team reviews the overall job and produces a high-level plan that covers all project phases and will produce all of the required deliverables.
5. In meeting 5, the team produces a quality plan. Here, the team members use the PSP quality measures to set numerical goals and agree on how to achieve them.
6. During meeting 6, the team members make detailed personal plans for the next several weeks. Here, each team member takes responsibility for a portion of the team's plan. Also during meeting 6, the team reviews every team member's workload and produces a balanced plan. Where one member has an excessive workload, others agree to take some tasks or to otherwise provide help.
7. In meeting 7, the team produces a project risk assessment and assigns team members to monitor and mitigate the key risks. The team thoughtfully

reviews the plan and identifies known and likely problems, quantifies their likelihood and impact, and agrees on the highest-priority mitigation efforts.

8. During meeting 8, the team prepares a plan presentation for management. If the plan does not meet management's desired goals, the team also produces one or more alternative plans. Although the team leader usually presents the plan, it is the entire team's plan and all of the members participate in its preparation and presentation.
9. In meeting 9, the team, team leader, and coach meet with the management group that attended meeting 1. Typically, the team leader presents the team's plan, and the team members help explain and defend it.
10. The final meeting is the postmortem during which the team reviews the launch process, records improvement suggestions, ensures that all important decisions and data are recorded, and assigns responsibility for any outstanding follow-up items.

With only one exception that I know of, management has always accepted the team's plan or one of its alternative plans. Out of many TSP team launches, the only exception was one in which management had asked for a one-year schedule and the team found that the work would take five years. Management then killed the project. This was really a success. Without the TSP, no one would have understood the magnitude of the job until after the team had spent a year or more trying to accomplish the impossible. The TSP saved the team a lot of frustration, and it saved management a lot of money.

14.6 The TSP Coach

Performing artists and competitive athletes know the value of coaching. Coaches help professionals to maximize their talents. Coaches in sports and the performing arts observe and study each practice and each performance. They are constantly looking for improvement opportunities. They push you to your known limits and then a little beyond. They can even drive you past what you thought was possible. They make the practice sessions so tough that the performance seems easy. Your coach is your greatest fan and your most demanding audience.

Accomplished coaches recognize talent and see the possibilities for excellence. As long as their charges start with the essential skills and a commitment to excellence, the coach will help them produce superior results. The objective is to make their occasional achievements seem normal and natural. Superior coaches also experiment. They watch each session and seek new ways to train and to motivate. They strive to help you improve while viewing every training session as a practice coaching session. It is unlikely that truly superior software development

performance will be achieved without the help of skilled coaches. The software coach has three objectives:

1. Motivate superior performance
2. Insist on a dedication to excellence
3. Support and guide individual and team development

With the TSP, coaches are first taught PSP skills and practices, and then trained in coaching. Organizations are urged to have at least one TSP coach for every four or five teams that they support. This gives the coaches enough time to both launch and support their teams and to spend time with each team and team member.

After an initial launch, new teams need full-time coaching support for at least two weeks. The members invariably have questions and are confused about how to handle common problems. Although experienced TSP teams usually need less coaching support, they do need periodic help and guidance to maintain their process discipline. Because the TSP represents a substantial change in working practice for most software developers, it is easy for them to get discouraged and to revert to their prior methods. Although many teams have been able to work through these issues by themselves, full-time coaching support can smooth TSP introduction and help to ensure team success.

14.7 Managing Your Own Project

Following the team launch and management's decision regarding the team's plan, the team follows its plan to do the work. In addition to the development work, the principal activities are managing the plan, tracking and reporting progress, managing change, and managing quality.

Maintaining the Plan

When new TSP teams first use the detailed plans they produced during the TSP launch, they invariably find that they have left out some important steps, included activities that are not essential, improperly defined some tasks, or made some significant estimating errors. Although this is normal, it can make the plan hard to follow. To handle these problems, teams must generally make many plan adjustments almost immediately after they start the job. Although these changes are usually relatively small and easy to make, it is important to keep the plan consistent with the way you work. If you don't, you will not be able to track your progress or see when you are falling behind. This makes it very difficult, if not impossible,

to make credible management status reports or to get sufficient warning of problems to take corrective action.

Another important part of maintaining the team plan is the team relaunch. Even though the overall team plan may extend for several years, the TSP's detailed plans extend for only a few months. Teams must therefore hold a relaunch every two to four months. A relaunch usually takes only two or three days, and it enables teams to update their plans based on what they have learned, to alert management and the customer to any problems, and to integrate any new members into the team.

Tracking and Reporting Progress

The role of development projects is to solve problems. We spend our lives dealing with unknowns and we frequently encounter unexpected problems and opportunities. Unfortunately, it seems to be a law of product development that every surprise involves more work. This means that our projects will always run late unless we recognize the problems in time to do something about them. For example, on a 26-week project, if you discover in week 5 that the work is taking longer than planned, you have time to replan the work, to get help, or to renegotiate the product's content. However, if you cannot precisely measure job status, you will not know that you are in trouble until it is too late to recover.

The TSP uses earned value (EV) to measure your status against the plan. With the help of this measure, you will always know job status to within a few hours. However, EV works only when the plan accurately represents what you are doing and when all team members regularly measure their work. Figure 14.1 shows an example of the status information that TSP teams get every week. This team had completed 7 weeks of a 17-week project and was falling behind. To understand the problem, the team planning manager made the following calculations:

1. The team members had spent 493.4 hours so far compared to a plan of 467 hours, so they had spent 5% more hours on the job than planned.
2. The team had accomplished 22.3 EV of work to date, compared to a planned 28.2 PV. This was a 26% slip in the schedule.
3. For the tasks the team had completed, the members had spent 458 hours compared to a planned 354.3 hours, or a 23% underestimate.
4. At the rate of 22.3 EV in 7 weeks, the team was earning 3.186 EV per week.
5. At this rate, the team would reach 100 EV in (100 – 22.3)/3.186 = 24.39 weeks.
6. Therefore, at the average EV rate the team had earned so far, it would take 24.39 + 7 = 31.39 weeks to complete the 17-week project, putting them 14.39 weeks behind schedule.

Weekly Data	Plan	Actual	Plan/ Actual
Schedule hours for this week	121.0	126.7	0.95
Schedule hours this cycle to date	467.0	493.4	0.95
Earned value for this week	7.6	6.4	1.19
Earned value this cycle to date	28.2	22.3	1.26
To-date hours for tasks completed	354.3	458.0	0.77
To-date average hours per week			

FIGURE 14.1 TSP TEAM WEEKLY STATUS (WEEK 7 OF A 17-WEEK JOB)

This summary form represents a real TSP team, and the members initially detected their schedule problem in week 3. They decided to concentrate on their task hours and to defer or stop any nonessential activities. Doing this increased their total task hours from an average of 70.5 per week to 126.7 hours in the latest week. If they could maintain this rate, they would be only 2.6 weeks late. To determine this, they made the following calculations:

1. At 458.0 hours for 22.3 EV, each EV point has been taking 20.54 hours.
2. At this rate, the hours required to reach 100 EV were 77.7 * 20.54 = 1,596 hours.
3. At the rate of 126.7 hours per week, they would reach 100 EV in 12.6 more weeks.
4. Because they had already spent 7 weeks on the job, they would finish in 19.6 weeks instead of the 17-week commitment, or 2.6 weeks late.

With this information, the team could monitor their work every day. In fact, the members continued to manage and even slightly increase their task hours and finished on exactly the day they had committed to management. With the TSP, teams can identify their problems early enough to handle them. That is why these teams generally deliver on or very close to their committed schedules.

Managing Change

It is "common knowledge" that the best way to develop software is to start with firm, clear, and complete requirements. However, experience shows that even when we get what is purported to be a firm requirement, it will change. The one exception I remember was when I was director of IBM's software development operations. I was so convinced that we should start with firm requirements that I threatened to cancel a big project if either the team or the marketing group changed the requirements after we had agreed on them. The product was then

produced on schedule and within costs, but it did not meet the customers' true needs. We then had to withdraw it and start over to develop the "right" product. This product failure was more expensive than the likely schedule delays that would have resulted from requirements changes.

Software developers soon learn that there is no such thing as a "firm requirement." Conversely, requirements changes can be very expensive. The key is to ensure that management understands the costs of every change as well as the cost of evaluating every proposed change. Then they will ask you to evaluate only the most critical changes. The TSP guidelines for handling the requirements-stability problem are as follows:

- Focus on quality from the beginning of the job. After the initial launch, the TSP process recommends that teams thoroughly review their requirements and resolve any questions, issues, or problems. This early focus on quality reviews, quality management, and quality planning will help you to identify requirements problems early, when they are least expensive to fix. Although most teams readily agree to doing this, they almost always seriously underestimate the time it will take to do it.
- Analyze the cost of every change, particularly the smallest ones. The big changes are generally easy to manage, but many small changes can nibble you to death. The PSP's detailed planning methods help you to quickly determine the impact of even the smallest change.
- Before accepting a change, make sure that the customer and management understand and agree to its benefits and expected impact. Again, the PSP planning process helps here.
- Because there is often disagreement about the meaning of a requirements statement, focus on the costs of the work and how your new understanding of the requirements impacts the plan. If the customer's interpretation is correct, as it often is, and if your plan was based on different assumptions, make sure that everyone understands the change's impact on the plan. Even if your project will have to "eat" the added costs, get the issue on the table as soon as you understand it and let management handle the cost and schedule problems with the customer. Again, the PSP planning methods will help you with this strategy.

TSP Quality Management

If the product didn't have to work, we could build it really quickly. Although it may not appear that the customers and managers care as much about quality as they do about cost and schedule, they do want our products to work. However, because quality is the key to both schedule management and product performance, quality is really our job. In fact, before they learn the PSP, most software devel-

opers spend 20% to 30% of their time finding and fixing defects. Then their teams spend nearly half of the project schedule finding and fixing more defects in testing. Finally, after delivery, many organizations spend as much money fixing defective products as they did originally developing them. By using the quality methods you learn with the PSP, you can cut these personal, team, and organizational costs by five to ten times. Furthermore, while you are doing this, you will produce more competitive products on shorter schedules and at less cost.

14.8 TSP Results

Although the PSP and TSP are relatively new, enough teams have used them to demonstrate their effectiveness. When the members are all trained, the team is professionally coached, and management provides proper leadership and support, TSP teams have been remarkably successful. In a study of 20 TSP projects in 13 organizations, schedule performance ranged from 20% ahead of schedule to 27% late. The average delay was 6% (Davis and Mullaney 2003). Although this is not perfect, it is far better than typical industry performance, where most projects are either cancelled or over 100% late.

The quality of these TSP teams' products ranged from 10 to 100 times better than previously, with an average of 60 defects per million lines of code. Over one-third of these products have had no customer-reported defects. These teams accomplished all of this while achieving an average 78% productivity improvement.

Although the business benefits of the TSP are impressive, the most significant improvement is in the developers' working environment. TSP team members find that their work is more rewarding and that they truly enjoy being on a cohesive and supportive team. With the TSP, team members typically report that software development is the fun that it ought to be.

14.9 The Rewards of Teamwork

After PSP training, quality work will seem normal and natural. In fact, TSP teams typically find that they don't want to revert to developing software the old way; it's much less fun and not nearly as satisfying. Learning and using the PSP will change your life. To understand why this is true, consider the factors that developers consider most important for project success. Linberg (Linberg 1999) has published a study of software developers' views of project success and failure, and he concludes that the following four points are most important:

1. A sense of being involved and making a significant contribution
2. An enjoyable and fun environment where even the little successes are recognized and celebrated with your managers and peers
3. Positive feedback from management and the customer on the product and the way you developed it
4. The autonomy to do the job in the way you think best

This is exactly the environment the TSP provides, and the following reactions of developers on TSP teams reflects their positive feelings:

- This really feels like a tight team.
- The TSP forces you to design, to think the whole thing out.
- Design time is way up but code time decreased to compensate.
- Tracking your time is an eye-opener.
- Really good teamwork on this project—no duplication of effort.
- I'm more productive.
- Gives you incredible insight into project performance.
- Wonderful to have team members assigned to specific roles.
- Team really came together to make the plan.
- I feel included and empowered.

14.10 The TSP Team of One

Whether you work on a TSP team or on a project by yourself, the PSP can help you to do more professional work and have a more rewarding career. To do so, however, you must consistently act like a professional software developer. Although this is demanding, it is also a much more enjoyable way to work. The key is to do the following:

- Plan and manage your personal work.
- Use effective personal methods.
- Recognize your strengths and weaknesses.
- Practice, practice, practice.
- Learn from history.
- Find and learn new methods.

Plan and Manage Your Personal Work

Whether you work alone or on a team, continue to use the PSP methods to plan your work, track your performance, and measure and manage the quality of the products you produce. When you are asked to do a job, take the time to make a plan before agreeing to a schedule. Next, keep your management informed of your progress at least every week. Then, whether or not you are on a self-directed team, you can be a self-directed software developer.

After PSP training, one developer was given a one-week assignment by his manager. Instead of accepting the job on the spot, he asked for a little time to make a plan. When the manager agreed, the developer estimated the size of the product and, from his PSP data, estimated that the work would take two weeks. When he showed his boss the detailed plan and explained why he needed more time, the manager agreed to the two-week schedule. This had never happened to the developer before and he was so excited that he called his coach to tell him. If you want to manage your own work, make plans for every job and negotiate these plans with your management. Be sure to base your plans on careful estimates, and base your estimates on your own historical data. Then, when management agrees to your plan, regularly track and report on your work.

Use Effective Personal Methods

The tragedy of modern software engineering is that almost all disasters are avoidable. Good software practices are straightforward, but few people consistently use them. Often, in fact, the appropriate methods were known and even planned for but not used. When the schedule pressures grow, as they always do, software developers often panic and throw out all of their good intentions in a mad scramble to meet the schedule. Unfortunately, this causes most software developers to immediately start coding and testing. Until they have used sound methods and seen the benefits, many professionals will claim that there is no evidence that these methods would really help them. In fact, there is plenty of evidence. As a dedicated professional, you must find the methods that work best for you and then use them. Finding them can take a little effort, but the really challenging part is using them once you find them.

Recognize Your Strengths and Weaknesses

Recognizing your strengths and weaknesses is the role of the postmortem. As you better understand your abilities, you will find tasks that you are good at and areas for improvement. Strengths and weaknesses are relative. Balance your

capabilities with those of your associates. Capitalize on your strengths and seek support where you are weak, and don't forget to ask for help. That is what teams are for. Teambuilding requires that you help your teammates as well as rely on them when you need help.

Treat every part of every project as a chance to learn. Measure your work, analyze the data, and see what works. Learn from variation. When things go well, figure out why. Practice what you did and attempt to repeat it. Perhaps you will develop a new trick or technique. Above all, focus on small improvements. If you can regularly make small improvements, the major changes will take care of themselves.

Practice, Practice, Practice

To improve your skills, get in the habit of practicing. Try new methods and work on personal problem areas. There may not be time for practice projects, but there are practice opportunities on most projects. You may need a prototype to prove a design or to resolve an interface. Prototypes are not always needed, but experiment when you can. Although you should use proven methods on product code, learn from your prototypes.

You can also practice on your projects, particularly those parts that are not on critical schedules. When learning a new method, practice it to build skill. Using new measurements, new process forms, or new analyses or verification methods is acceptable, as long as it does not jeopardize the project's quality and schedule performance. Practice is an essential part of personal improvement, so make practicing a normal part of your daily work.

Learn from History

The difference between a tragedy and a fiasco is that in a tragedy, you have not learned from the fiasco. We do few things for the first time. Invariably, somebody has produced a program just like the one you are now developing. They have made the mistakes you will make, and they know the traps and pitfalls. A little exploration can save you a lot of grief. Look for relevant experience and check the literature. You will often find pertinent books and articles.

As Santayana once observed, "Those who cannot remember the past are condemned to repeat it" (Santayana 1905). Observe and learn from others and, as you build your knowledge, make it known to others. You can then be part of building a more effective software engineering discipline. The power of science comes from the accumulating wealth of knowledge. Take advantage of what you find and contribute what you have learned.

Find and Learn New Methods

This book has introduced a number of methods and techniques, but they are only examples. Many tools and methods are now available and new ones are constantly being developed. You cannot learn everything, but you can watch for innovations that are pertinent to your personal needs. Keep a list and select those that are most important to you now. Allocate time for skill-building and spend a few hours each week learning something new. If you do this consistently, you will learn the equivalent of one graduate course per year. With the current pace of technology, this is the bare minimum needed to keep up.

The rate of technical change is accelerating and new tools and methods are being developed faster than ever before. Although there is a vast amount of literature to monitor, many tools are available to help us. Most libraries will run keyword searches and provide lists of pertinent articles. The Internet is a vast storehouse of potentially useful information. Think of it this way: Suppose you had a serious medical problem and wanted a competent doctor. Would you pick one who never cracked a journal or attended a conference? Or would you seek a skilled person who regularly updates his or her skills? When you trust your life to someone, you want them to behave professionally.

Your professional life is no different. Should you entrust it to someone who is out-of-date? If you were looking for career guidance, would you ask someone who had not studied a technical book since college? Probably not. The skill, knowledge, and ability you bring to your job will determine your future. Your future is in your hands.

14.11 Your Future in Software Engineering

As you look to the future, you will face many questions. How will your field evolve and what can you do to meet the mounting challenges? Although no one can know the future, your progress will likely be limited by your ability to build personal skill. Make practice a part of every project, and measure and observe your own work. Since you cannot stand still, treat every project as a way to build talent, rather than merely treating your talent as a way to build projects.

Deciding what you want from your chosen field is like asking what you want from life. Surprisingly often, people achieve their objectives, but in ways they did not expect. Life rarely turns out the way we plan. Although our carefully developed strategies may go down in flames, a new and more rewarding opportunity often shows up in the ashes. The key is to keep an open mind and to keep looking. In life, we all reach the same end, so we need to concentrate on the trip. Just as with a process, once you decide how you want to live, the rest will follow. Devote yourself to excellence and you just might achieve it. That would be worth the trip.

References

Davis, N., and J. Mullaney. "Team Software Process (TSP) in Practice." SEI Technical Report CMU/SEI-2003-TR-014, September 2003.

Dyer, J. L. "Team Research and Team Training: A State-of-the-Art Review." *Human Factors Review* (1984): 286, 309.

Humphrey, W. S. *Winning with Software: An Executive Strategy.* Boston: Addison-Wesley, 2002.

Linberg, K. R. "Software Developer Perceptions about Software Project Failure: A Case Study." *Journal of Systems and Software* 49 (1999): 177–192.

Martin, D. *Team Think: Using the Sports Connection to Develop, Motivate and Manage a Winning Business Team.* New York: Dutton, 1993.

Santayana, G. *The Life of Reason.* (1905). (Reprinted by Dover Publications, 1982, p. 284.)

Standish Group. "CHAOS: A Recipe for Success." Dennis, MA: The Standish Group International, Inc., 1999. www.standishgroup.com.